DATADVANCE
AN EADS COMPANY

MACROS

GTApprox

Generic Tool for Approximation

© 2007 — 2013 DATADVANCE, llc

Contact information

Phone	+7 (495) 781 60 88	
Web	`www.datadvance.net`	
Email	`support@datadvance.net`	Technical support, questions, bug reports
	`info@datadvance.net`	Everything else
Mail	DATADVANCE, llc Pokrovsky blvd. 3, building 1B 109028 Moscow Russia	

User manual prepared by: D. Yarotsky (principal author), M. Belyaev, E. Krymova, A. Zaytsev, Yu. Yanovich, E. Burnaev

Contents

List of figures **v**

List of tables **vii**

1 Introduction **1**
1.1 What is GT Approx . . . 1
1.2 Documentation structure . . . 1

2 Overview **3**

3 The workflow **8**
3.1 Overview . . . 8
3.2 Data processing . . . 8

4 Approximation techniques **10**
4.1 Individual approximation techniques . . . 10
4.1.1 Regularized linear regression . . . 10
4.1.2 1D Splines with tension . . . 10
4.1.3 High Dimensional Approximation . . . 11
4.1.4 Gaussian Processes . . . 16
4.1.5 High Dimensional Approximation combined with Gaussian Processes 19
4.1.6 Sparse Gaussian Process . . . 20
4.1.7 Response Surface Model . . . 21
4.2 Limitations . . . 23
4.3 Example: comparison of the techniques on a 1D problem . . . 24

5 Selection of approximation technique **26**
5.1 Manual selection . . . 26
5.2 Automatic selection . . . 26

6 Special approximation modes **30**
6.1 Linear mode . . . 30
6.2 Interpolation mode . . . 31
6.3 Componentwise training of the approximation . . . 32
6.4 Noisy problems . . . 33
6.5 Heteroscedastic data . . . 34
6.6 Data with errorbars . . . 35
6.7 Smoothing . . . 36

7 Control over accuracy **38**
7.1 Overview of accuracy estimation and related functionality of the tool 38
7.2 Selection of the training data 40
7.3 Limitations of accuracy estimation 40

8 Control over training time **44**
8.1 General notes 44
8.2 Accelerator 45

9 Accuracy Evaluation **48**
9.1 Overview 48
9.2 Usage examples 50
9.2.1 AE with interpolation turned on or off 50
9.2.2 An adaptive DoE selection 51

10 Internal Validation **57**
10.1 Description of the procedure 57
10.2 Notes and Remarks 58
10.3 Usage examples 61
10.3.1 Internal Validation for a simple and a complex functions 61
10.3.2 Comparison of Internal Validation and validation on a separate test set 61
10.3.3 Scatter plot obtained with IV predictions 64

11 Processing user-provided information **65**
11.1 Preprocessing 65
11.1.1 Analysis step 66
11.1.2 Processing user responses 66
11.1.3 Example 67
11.2 Postprocessing 67
11.2.1 Postprocessing analysis 67
11.2.2 User's opinion treatment 68
11.2.3 Example 68

12 Tensor products of approximations **73**
12.1 Introduction 73
12.2 Discrete/categorical variables 76
12.3 The overall workflow 77
12.4 Factorization of the training set 78
12.5 Approximation in single factors 80
12.6 Smoothing 82
12.7 Difference between TA and iTA techniques 82
12.8 Examples 82

13 Multi-core scalability **84**
13.1 Parallelization 84
13.2 Hyperthreading 84
13.3 Recommendations 85

A Details of approximation techniques **87**
A.1 LR . . . 87
A.2 SPLT . . . 88
A.2.1 The case of one-dimensional output. . . . 88
A.2.2 The case of multidimensional output . . . 89
A.3 HDA . . . 90
A.3.1 Elementary Approximator Model . . . 90
A.3.2 Training of Elementary Approximator Model . . . 91
A.3.3 Options of HDA and their mathematical counterparts . . . 91
A.4 GP . . . 92
A.4.1 Used class of GPs . . . 92
A.4.2 Tuning of covariance function parameters . . . 93
A.4.3 Options of GP and their mathematical counterparts . . . 93
A.4.4 Sparse GPs . . . 94
A.4.5 GP with errorbars . . . 94
A.5 HDAGP . . . 95
A.6 RSM . . . 96
A.6.1 Parameters estimation . . . 96
A.6.2 Categorical variables . . . 98

References **99**

Acronyms **102**

Index: Concepts **104**

Index: Options **105**

List of Figures

2.1 Examples of approximations 6
2.2 Workflow comparing different approaches to selection of next initial point . . 7

4.1 Comparison of different methods 25

5.1 The internal decision tree 27
5.2 The "sample size vs. dimension" diagram of default techniques 29

6.1 Examples of approximations with linear mode OFF or ON 31
6.2 Examples of approximations with Interpolation OFF or ON 32
6.3 Different approximation techniques on a noisy problem 34
6.4 The results of GP technique, ordinary vs. heteroscedastic version 35
6.5 2D smoothing 37
6.6 1D smoothing 37

7.1 Uniform and non-uniform DoE 41
7.2 Uniform and random DOE in 1D 41
7.3 A misleading training set 42

9.1 An example of accuracy prediction 49
9.2 AE predictions with interpolation on or off 51
9.3 Adaptive DoE: the approximated function f 52
9.4 Adaptive DoE, initial state 53
9.5 Adaptive DoE, final state 55
9.6 Adaptive DoE, error vs size of the training set 56

10.1 Rosenbrock and Rastrigin functions 62
10.2 Comparison of Internal Validation and validation on a separate test set . . . 63
10.3 Comparison of scatter plots obtained with Internal validation and a separate test sample 64

11.1 True function 70
11.2 Approximation for the surrogate model constructed using default options . . 70
11.3 Approximation for the surrogate model constructed using recommended by postprocessing function options 71
11.4 Scatter plot for the training sample for the wine quality problem 71

12.1 Examples of full factored sets 74
12.2 Examples of incomplete factored sets 74
12.3 The overall workflow with tensored approximations 79

12.4 Decision tree for choosing approximation technique in a single factor 81
12.5 Examples of tensored approximations. 83

13.1 Multi-core scalability 85

List of Tables

4.1 Limitations of approximation techniques 24

6.1 Noise–sensitivity of different approximation techniques 33

8.1 Effect of `Accelerator` on training time and accuracy 47

10.1 Internal Validation results for the Rosenbrock and Rastrigin functions 62

11.1 Comparison of errors on the test set 72

A.1 RSM types 96
A.2 Pre-coding with dummy variables 98

Chapter 1

Introduction

1.1 What is GT Approx

GTApprox is a software package for construction of approximations fitting user-provided training data and for assessment of quality of the constructed approximations.

1.2 Documentation structure

This User Manual includes:

- A general overview of the tool's functionality;
- Short descriptions of the algorithms;
- Recommendations on the tool's usage;
- Examples of applications to model problems.

In addition to this User Manual, documentation for **GTApprox** includes *Technical References*, which cover the programming-related aspects of the tool's usage. In particular, they contain:

- A full detailed list of available functions and options;
- Excerpts of code illustrating the tool's usage.

The present User Manual has the following structure:

- Chapter 2 is an introduction to the tool and the relevant approximation concepts.
- Chapter 3 describes the internal workflow of the tool.
- Chapter 4 describes specific approximation techniques implemented in the tool.
- Chapter 5 describes how the approximation technique is selected in a particular problem.
- Chapter 6 describes a number of special approximation modes (linear, interpolating, noisy, separate/joint training) which may be useful in some applications, and explains how they are realized in the tool.

- Chapter 7 is a general introduction to the accuracy–related topics relevant for the tool.
- Chapter 8 describes methods of tuning the training time.
- Chapter 9 describes the functionality of the tool known as *Accuracy Evaluation* and examples of its applications.
- Chapter 10 describes the functionality of the tool known as *Internal Validation* and examples of its applications.
- Chapter 12 describes how to construct tensor products of approximations with the tool.
- Appendix A contains mathematical details of approximation techniques implemented in the tool.

Chapter 2

Overview

The main goal of Generic Tool for Approximation (**GT Approx**) is to construct *approximations*[1] fitting the user-provided *training sets* [2].

A training set S is a collection of vectors representing an unknown *response function*

$$Y = f(X).$$

Here, X is a d_{in}-dimensional vector, and Y is d_{out}–dimensional. In general, dimensions d_{in}, d_{out} can be greater than 1.

A single element of the training set is a pair (X_k, Y_k), where $Y_k = f(X_k)$. We denote the total number of elements in the training set — the *size of the training set* — by $|S|$.

Sometimes, it is necessary to refer to the set $\{X_k\}_{k=1}^{|S|}$ of the input parts of training vectors. We refer to this set as the *Design of Experiment (DoE)*.

The training set S is usually numerically represented by a $|S| \times (d_{\text{in}} + d_{\text{out}})$–matrix $(\mathbf{XY})_{\text{train}}$. We can think of this matrix as having $|S|$ rows, corresponding to different training vectors, and $d_{\text{in}} + d_{\text{out}}$ columns, corresponding to different scalar components of these vectors (d_{in} components of the input and d_{out} of the output). This matrix naturally divides into two submatrices — $\mathbf{X}_{\text{train}}$, corresponding to the DoE, and $\mathbf{Y}_{\text{train}}$, corresponding to the output components of the training set.

Given a training set S, **GTApprox** constructs an approximation $\widehat{f}(\equiv f_{\text{approx}})$,

$$\widehat{f} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}},$$

to the response function f. The tool attempts to find an optimal approximation, providing a good fit to training data yet reasonably simple so as to maintain the predictive power of the approximation. See Section 7 for a brief discussion of issues related to accuracy and predictive power of the approximation.

If noise level in output values from the training set is available then these values can be used as another input of **GTApprox**. In this case algorithm works with extended training set $(X_k, Y_k, \Sigma_k)_{k=1}^{|S|}$, where $\Sigma_k = \Sigma(X_k) \in \mathbb{R}^{d_{\text{out}}}$ is a variance of noise in output components at point X_k. The knowledge of noise variances allows algorithm to provide better approximation quality for noisy problems (See Section 6.6 for details).

There are different approaches to definition and construction of optimal approximations (see, e.g., [19]), and **GTApprox** combines several such approaches in an efficient and convenient-to-use form.

[1] also known as *surrogate models*, *meta-models* or *response surfaces*, see, e.g., [19]

[2] also known as *training data* (or *samples*)

First, **GTApprox** contains several different approximation algorithms, which in the sequel are referred to as *approximation techniques*. Each of these techniques has its merits, and is very efficient for some application conditions. See Section 4 for descriptions of these techniques.

An important property of **GTApprox** is that, by default, for any given problem the tool automatically selects an appropriate technique. The choice is determined by the properties of the training data and the user–defined options. The selection algorithm is outlined in Section 5. Some examples of approximations constructed by **GTApprox** in the default mode are shown in Figure 2.1.

If the approximation goes through the training points, i.e., $\widehat{f}(X_k) = Y_k$, it is referred to as *interpolation*. In general, approximations created by **GTApprox** are not interpolating, but the tool has an option enforcing this property; see Section 6.2. Note that an interpolating approximation is not necessarily the most accurate; see Section 7 for a discussion.

In addition to interpolation, **GTApprox** supports a number of other special approximation modes, see Section 6. For example, **GTApprox** can construct an optimal linear approximation. Also, noise–filtering capabilities of different approximation techniques of **GTApprox** are discussed in Section 6.4.

The approximation $\widehat{f}$ produced by **GTApprox** is defined for all $X \in \mathbb{R}^{d_{\text{in}}}$, but its accuracy normally deteriorates away from the training set. In practice, the approximation is usually considered on a bounded subset D of $\mathbb{R}^{d_{\text{in}}}$ known as *design space*. A number of issues related to controlling the accuracy of the approximation, including general methodology of accuracy measurements and the appropriate choice of training set, are discussed in Section 7.

In some cases, it may be desirable to reduce the training time of the approximation at the cost of making it somewhat less accurate, or, conversely, obtain a more accurate approximation by letting it train for longer. This is discussed in Section 8.1.

In addition to constructing the approximation, **GTApprox** provides extra functionality related to this approximation and expanding the content of the created model:

- The tool optionally applies additional *smoothing* to the constructed approximation. See Section 6.7.
- The constructed approximation can be exported to the human-readable **Octave** format.
- The tool provides the gradient (Jacobian matrix) of the approximation, $\nabla\widehat{f} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$.
- The approximation is optionally supplemented with an accuracy prediction $\sigma : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}_{+}$ — a function which, for any given design point X, estimates the accuracy of the approximation at this point. This functionality goes under the name *Accuracy Evaluation* and is described in Section 9.
- If *Accuracy Evaluation* is available, the tool also provides the gradient (Jacobian matrix) of accuracy prediction, $\nabla\sigma : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$.
- The tool optionally performs *Internal Validation* of the selected approximation technique on the training set. This procedure consists in a generalized cross–validation of this technique on the supplied training data. The output of this procedure is a set of errors providing the user with an estimate of the overall accuracy of the approximation. See Section 10 for more details.

The tool does not assume any specific structure of data and is primarily intended for generic *scattered* training sets. However, the tool contains a complex functionality of forming *tensor products of approximations* for data having the structure of a *Cartesian product* or an *incomplete Cartesian product*, which is very common in practice. This is described in Section 12.

The overall usage pattern of **GTApprox** is shown in Figure 2.2. It includes two stages:

- *construction*[3] of the approximation (Figure 2.2a);
- *evaluation*[4] of the constructed approximation (Figure 2.2b).

To construct the approximation, the user supplies the tool with training data and, possibly, specifies some of its *options*. The tool then produces a *model* comprising the approximation and some additional data related to it. In particular, these data include Internal Validation errors if they were requested by the user. The approximation is internally piecewise represented by analytic expressions which can be exported to a text file.

The constructed model can then be evaluated on a vector from the design space. The model outputs the values of the approximation and its gradient at this vector. If AE is available, model also outputs the accuracy prediction and its gradient. One can optionally smooth the approximation before evaluating it.

Options of **GTApprox** can be divided into two groups:

- *Basic* options control either general user-specified properties of the approximation or the additional functionality provided with the approximation. In **GTApprox**, this group contains four ON/OFF switches[5]:

  Lin: Require the approximation to be a linear function. See Section 6.1.

  ExFit: Require the approximation to fit training data exactly. See Section 6.2.

  Int: Same as **ExFit** (exact interpolation mode).

  AE: Require the constructed model to contain the accuracy prediction. See Section 9.

  IV: Require Internal Validation to be performed during the model construction. See Section 10.

  Also, this group contains **Accelerator** taking values 1 to 5. It allows the user to adjust the training time, see Section 8.2.

- *Advanced* options control more technical aspects of individual algorithms and can be used to specify the approximation technique or fine-tune the algorithms; see Sections 4 and 5.

Approximations constructed by **GTApprox** are continuous and piece-wise analytic functions. The gradients of the approximations and the error predictions (or Jacobi matrices, in case $d_{\text{out}} > 1$) are obtained by analytic differentiation, which ensures their accuracy.

See Technical reference [15] for implementation details.

[3] *construction* is synonymous to *training* and *building*

[4] *evaluation* is synonymous to *simulation*

[5] For brevity, throughout this manual we mostly use short versions of options' names. For example, the full name of the option for interpolation, as implemented in the software, is `GTApprox/InterpolationRequired`. We always omit the prefix `GTApprox/`, common to all training options of **GTApprox**. Furthermore, we shorten `LinearrityRequired`, `InterpolationRequired`, `AccuracyEvaluation`, `InternalValidation` to **Lin**, **ExFit** (**Int**), **AE**, **IV**, respectively.

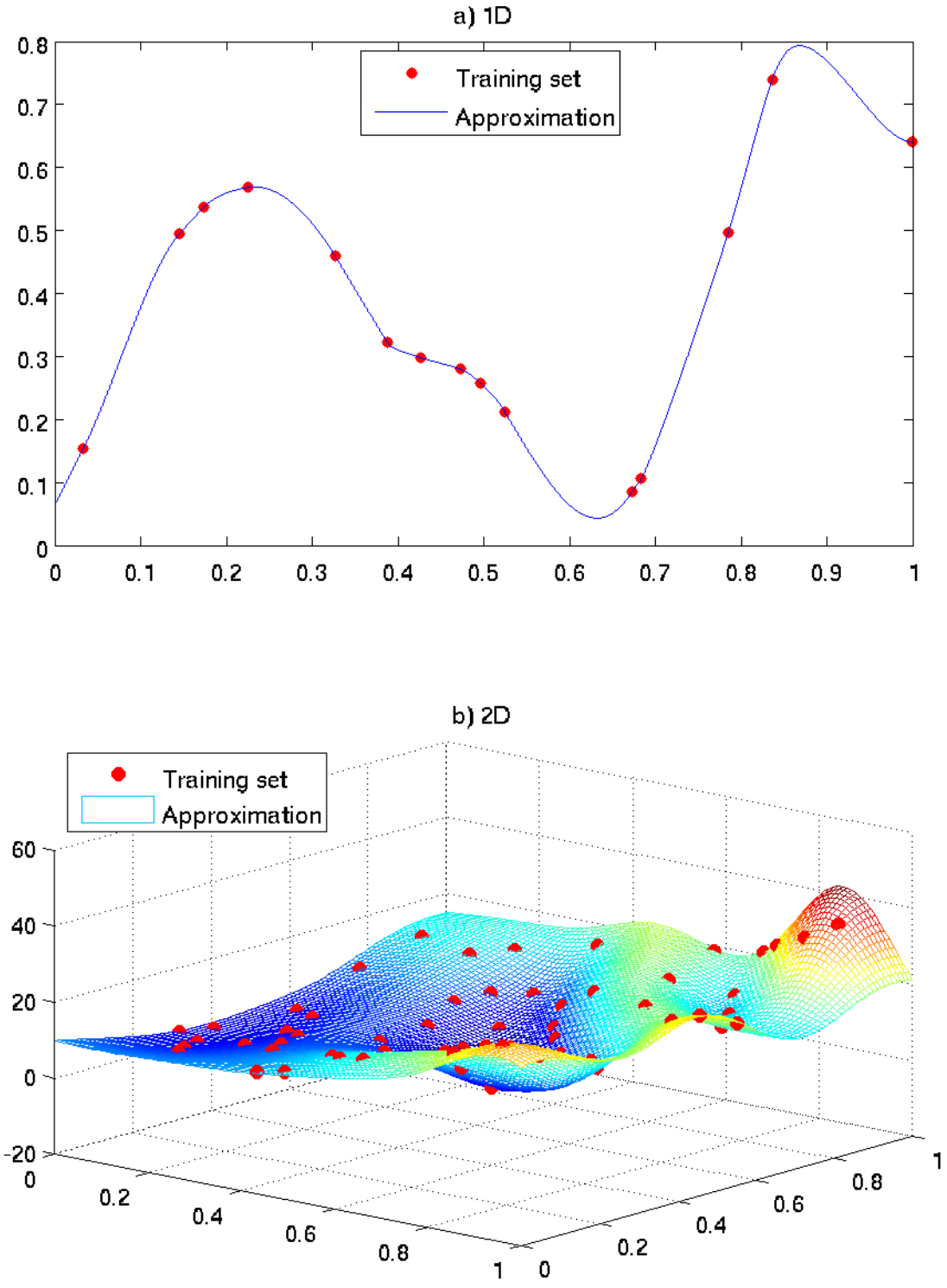
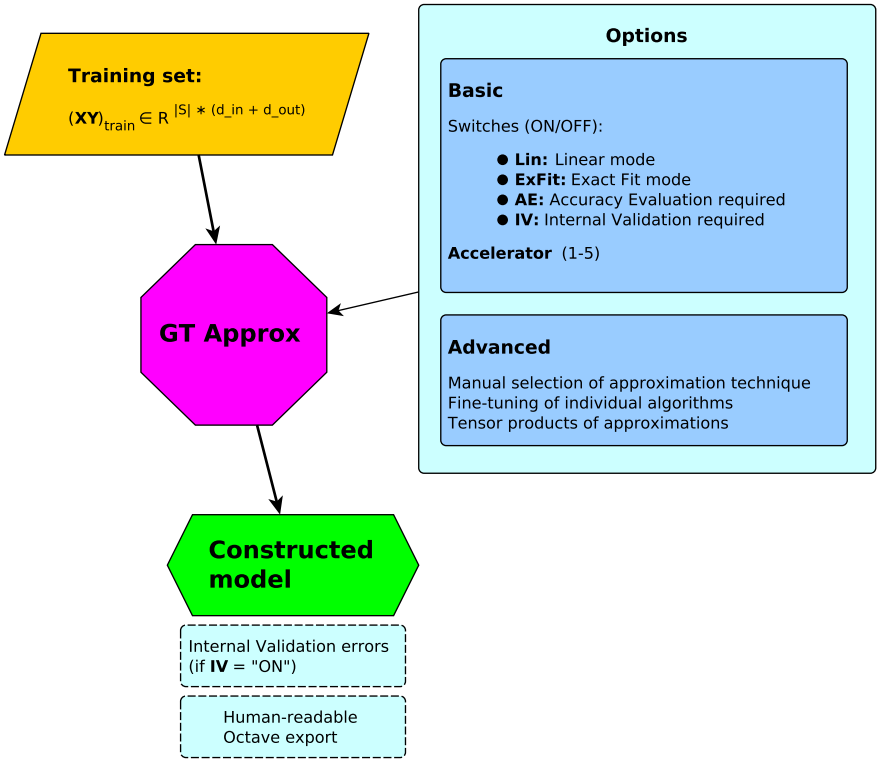
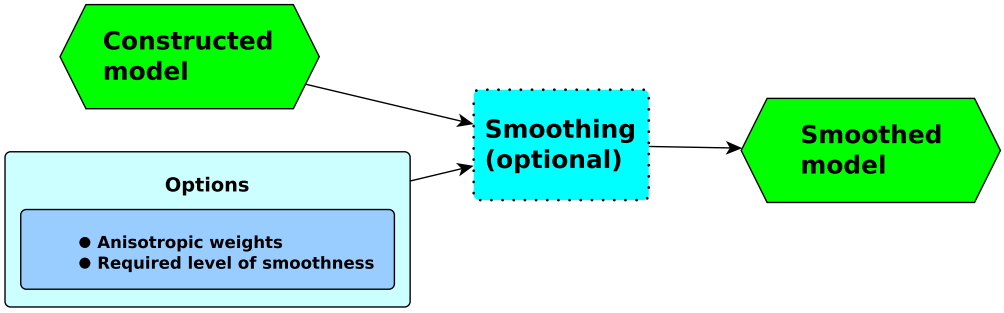


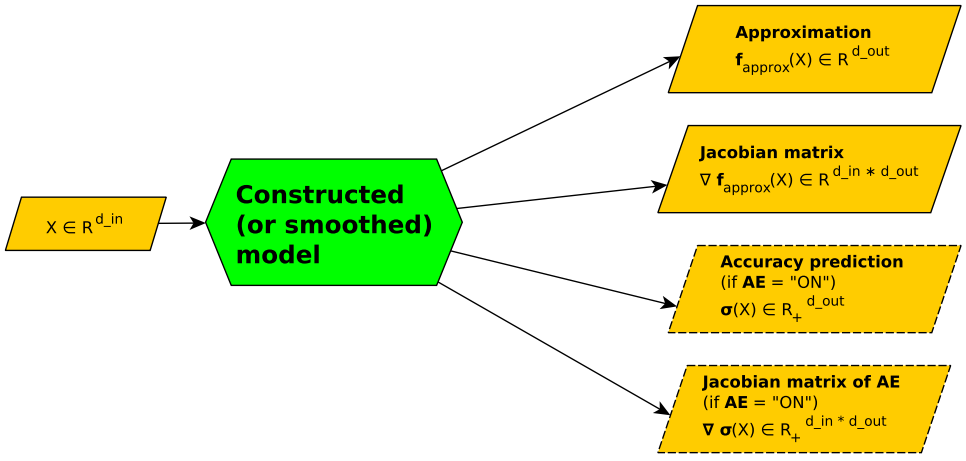
Figure 2.1: Examples of approximations constructed by **GTApprox**: a) $d_{\text{in}} = 1$, $d_{\text{out}} = 1$, $|S| = 15$; b) $d_{\text{in}} = 2$, $d_{\text{out}} = 1$, $|S| = 60$
Note: This image was obtained using an older MACROS version. Actual results in the current version may differ.



(a) Construction of the model



(b) Smoothing of the constructed model (optional)


Approximation
f approx(X) ∈ R d_out
Jacobian matrix
∇ f approx(X) ∈ R d_in ∗ d_out
X ∈ R d_in
Constructed (or smoothed) model
Accuracy prediction
(if AE = "ON")
σ(X) ∈ R+ d_out
Jacobian matrix of AE
(if AE = "ON")
∇ σ(X) ∈ R+ d_in * d_out

(c) Evaluation of the constructed model

Figure 2.2: GTApprox workflow

Chapter 3

The workflow

3.1 Overview

The workflow with the tool includes the following elements.

- **Construction of the model.** Construction of the model internally consists of the following steps:

  1. **Preprocessing.** At this step, redundant data are removed from the training set. See Section 3.2.
  2. **Analysis of training data and options, selection of approximation technique.** At this step, the parameters of the training data and the user–specified options are analyzed for compatibility, and the most appropriate approximation technique is selected. See Section 5.
  3. **Internal Validation (optional).** At this step, Internal Validation of the selected approximation technique is performed on the training data, if the corresponding option is turned on. See Section 10.
  4. **Construction of the main approximation with (optionally) Accuracy Evaluation prediction.** At this step, the main approximation, to be returned to the user, is constructed. See section 4 for descriptions of individual approximation techniques. Also, the accompanying Accuracy Evaluation prediction is constructed if the corresponding option is turned on. See Section 9.

  After the model has been constructed, the user can view training settings, Internal Validation errors (if available) and export the approximation to a text file.

- **Smoothing of the model (optional).** User can smooth the model after training in order to decrease its variability. See Section 6.7

- **Evaluation of the model.** Evaluation of the constructed model on a point from the design space including the approximation, gradient of approximation, accuracy prediction and its gradient (if available).

3.2 Data processing

Before applying the approximation technique to the training set, this training set is preprocessed in order to remove possible degeneracies in the data. Let $\mathbf{XY}$ be the $|S| \times (d_{\text{in}} + d_{\text{out}})$

matrix of the training data, where the rows are $(d_{\text{in}} + d_{\text{out}})$-dimensional training points, and the columns are individual scalar components of the input or output. As explained in Section 2, the matrix $\mathbf{XY}$ consists of the sub-matrices $\mathbf{X}$ and $\mathbf{Y}$. We perform the following operation with the matrix $\mathbf{XY}$:

1. We remove all constant columns in the sub-matrix $\mathbf{X}$. A constant column in $\mathbf{X}$ means that all the training vectors have the same value of one of the input components. In particular, this means that the training DoE is degenerate and lies in a proper subspace of the design space. When constructing the approximation, such input components are ignored.

2. We remove repeating rows in the matrix XY. A repeating row means that the same training vector is included more than once in the training matrix. Repetitions bring no additional information and are therefore ignored; a repeating row is counted only once.

If the above operations are nontrivial, e.g., if the matrix $\mathbf{X}$ does contain constant columns or the matrix $\mathbf{XY}$ does contain repeating rows, then the removals are accompanied by warnings.

As a result of these operations, we obtain a reduced matrix $\widetilde{\mathbf{XY}}$ consisting of the submatrices $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{Y}}$. Accordingly, we define *effective input dimension* and the *effective sample size* as the corresponding dimensions of the sub-matrix $\widetilde{\mathbf{X}}$.

Note that after removing repeating rows in $\mathbf{XY}$ the reduced matrix may still contain rows which have the same X components but different Y components. This means that the training data is so noisy that, in general, several different outputs correspond to the same input. Such noisy problems may require a special tuning of **GTApprox**, see Section 6.4; in particular not all approximation techniques are appropriate for them. If the training data does contain rows with equal X but different Y components, the tool produces a warning.

The model constructed by **GTApprox** is constructed using the reduced matrix $\widetilde{\mathbf{XY}}$ rather than the original matrix $\mathbf{XY}$. Furthermore, it is the effective input dimension and sample size, rather than the original ones, that are used when required at certain steps, in particular when determining the default approximation technique (see Section 5) and choosing the default Internal Validation parameters (see Section 10). For example, if the original input dimension has been reduced to 1, then the default 1D technique will be applied to this data by default. The resulting approximation is then considered as a function of the full d_{in}-dimensional input, but it depends only on those components of the full input which have been included in $\widetilde{\mathbf{X}}$.

Chapter 4

Approximation techniques

GTApprox combines a number of approximation techniques of different types. By default, the tool selects the optimal technique compatible with the user–specified options and in agreement with the best practice experience. Alternatively, the user can directly specify the technique through advanced options of the tool. This section describes the available techniques; selection of the technique in a particular problem is described in Section 5.

4.1 Individual approximation techniques

The following sections describe general aspects of approximation techniques of **GTApprox**. For their mathematical details, see Appendix A.

4.1.1 Regularized linear regression

Short name: LR

General description: Regularized linear regression. Regularization coefficient is estimated by minimization of generalized cross-validation criterion [22].

Strengths and weaknesses: A very robust and fast technique with a wide applicability in terms of the space dimensions and amount of the training data. It is, however, usually rather crude, and the approximation can hardly be significantly improved by adding new training data. Also, see Section 6.1.

Restrictions: Interpolation mode is not supported, AE is not supported. Large samples and dimensions are fully supported (up to machine-imposed limits).

Options: No options.

4.1.2 1D Splines with tension

Short name: SPLT

General description: A one–dimensional spline–based technique intended to combine the robustness of linear splines with the smoothness of cubic splines. A non–linear algorithm is used for an adaptive selection of the optimal weights on each interval between neighboring points of DoE. The default implementation in **GTApprox** is already interpolating, so that the **Int** switch has no effect on it. See [23, 27, 28] for details.

Strengths and weaknesses: A very fast technique, combining robustness of linear splines with the accuracy and smoothness of cubic splines. Interpolating. Should not be applied to very noisy problems (see Figure 6.3). On the other hand, performs well if the training data is noiseless, see Figure 2.1a). Is the default method for 1D problems in **GTApprox**.

Restrictions: Only for 1D models ($d_{\text{in}} = 1$). Can be used with very large training sets (more than 10000 points).

Options:

- `SPLTContinuity`

  Values: enumeration: `C1`, `C2`

  Default: `C2`

  Short description: required approximation smoothness.

  Description: If `C2` is selected, then the approximation curve will have a continuous second derivative. If `C1` is selected, only the first derivative will be continuous.

4.1.3 High Dimensional Approximation

Short name: HDA

General description: A non–linear, self–organizing technique for construction of approximation using linear decomposition in functions from functional dictionary consisting of both linear and nonlinear base functions. Particular example of such decomposition is so-called two-layer perceptron with nonlinear (sigmoid) activation function. However, the structure and algorithm of HDA is completely different from that of two-layer perceptron and contains the following features:

- an advanced algorithm of division of the training set into the proper training and validating parts;
- different types of base functions, including sigmoid and Gaussian functions (see also description of options `HDAFDLinear`, `HDAFDSigmoid` and `HDAFDGauss` below);
- adaptive selection of the type and number of base functions and approximation's complexity (see also description of options `HDAPMin` and `HDAPMax` below);
- an innovative algorithm of initialization of base functions' parameters (see also description of options `HDAInitialization` and `HDARandomSeed` below);
- several different optimization algorithms used to adjust the parameters, including RPROP, Levenberg–Marquardt and their modifications;

- an adaptive strategy controlling the type and parameters of used optimization algorithms (see also description of the option `HDAHPMode` below);
- an adaptive regularization algorithm for controlling the generalization ability of the approximation;
- multi–start capabilities to globalize the optimization of the parameters (see also description of options `HDAMultiMin` and `HDAMultiMax` below);
- an advanced algorithm for boosting used for construction of ensembles for additional improvement of accuracy and stability (see also description of the option `HDAPhaseCount` below);
- post–processing of the results to remove redundant features in the approximation.

In short the HDA algorithm works as follows:

1. The training set is devised into the proper training and validating parts.
2. Functional dictionary with different types of base functions, including sigmoid and Gaussian functions, is initialized.
3. The number, type of base functions from the functional dictionary and approximation's complexity are adaptively selected. Thus a basic approximator is initialized.
4. The basic approximator is trained using an adaptive strategy controlling the type and parameters of used optimization algorithms. Specially elaborated adaptive regularization is used to control the generalization ability of the basic approximator.
5. Post–processing of the basic approximator's structure is done in order to remove redundant base functions.
6. An ensemble consisting of such basic approximators is constructed using an advanced algorithm for boosting and multi–start.

See [4] for details.

Strengths and weaknesses: HDA is a flexible nonlinear approximation technique, with a wide applicability in terms of space dimensions. HDA can be used with large training sets (more than 1000 points). However, HDA is not well-suited for the case of very small sample sizes. Significant training time is often necessary to obtain accurate approximation for large training samples. HDA can be applied to noisy data (see Section 6.4). Usually, accuracy of HDA approximations improves as more training data are supplied.

Restrictions: Interpolation mode is not supported, AE is not supported. Large samples and dimensions are supported.

Options:

- `HDAPhaseCount`

  Values: integer [1, 50].

  Default: 10.

  Short description: Maximum number of approximation phases.

  Description: PhaseCount parameter specifies the maximum possible number of approximation phases. The parameter can take positive integer values. Usually for particular problem the number of approximation phases completed by the approximation algorithm is smaller than the maximum possible number of approximation phases (approximation algorithm stops performing approximation phases as soon as the subsequent approximation phase do not increase the accuracy). In general the more approximation phases we have the more accurate approximator we get. However increase of the maximum possible number of approximation phases can lead to significant increase of training time or/and to overfitting in some cases.

- `HDAPMin`

  Values: integer [0, HDAPMax].

  Default: 0.

  Short description: Minimum admissible complexity.

  Description: Parameter specifies minimal admissible complexity of the approximator. This parameter can take non-negative integer values and must be less or equal to the value of the parameter HDAPMax. The approximation algorithm selects the approximator with optimal complexity pOpt from the range [HDAPMin, HDAPMax]. Optimality here means that depending on the complexity of the behavior of approximable function and the size of the available training sample the constructed approximator with complexity pOpt fits this function in the best possible way compared to approximators with other complexities from the range [HDAPMin, HDAPMax]. Thus the parameter HDAPMin should not be too big in order to select the approximator with complexity being the most appropriate for the considered problem. In general increase of the parameter HDAPMin can lead to significant increase of training time or/and to overfitting in some cases.

- `HDAPMax`

  Values: integer [HDAPMin, 5000] non-negative integer, greater than or equal to HDAPMin.

  Default: 150.

  Short description: Maximum admissible complexity.

  Description: Parameter specifies maximal admissible complexity of the approximator. This parameter can take non-negative integer values and must be greater or equal to the value of the parameter HDAPMin. The approximation algorithm selects the approximator with optimal complexity pOpt from the range [HDAPMin, HDAPMax]. Optimality here means that depending on the complexity of the behavior of approximable function and the size of the available training sample

the constructed approximator with complexity pOpt fits this function in the best possible way compared to approximators with other complexities from the range [HDAPMin, HDAPMax]. Thus the parameter HDAPMax should be big enough in order to select the approximator with complexity being the most appropriate for the considered problem. In general increase of the parameter HDAPMax can lead to significant increase of training time or/and to overfitting in some cases.

- `HDAMultiMin`

  Values: integer [1, HDAMultiMax]

  Default: 5.

  Short description: Minimal number of basic approximators constructed during one approximation phase.

  Description: Parameter specifies minimal number of basic approximators constructed during one approximation phase. The parameter can take positive integer values and must be less or equal to the value of the parameter HDAMultiMax. In general the bigger the value of HDAMultiMin the more accurate approximator we get. However increase of this parameter can lead to significant increase of training time or/and to overfitting in some cases.

- `HDAMultiMax`

  Values: integer [HDAMultiMin, 1000].

  Default: 10.

  Short description: Maximal number of basic approximators constructed during one approximation phase.

  Description: Parameter specifies maximal number of basic approximators constructed during one approximation phase. The parameter can take positive integer values and is greater or equal to the value of the parameter HDAMultiMin. Usually for particular problem the number of basic approximators constructed by the approximation algorithm is smaller than the maximum possible number HDAMultiMax of basic approximators (approximation algorithm stops constructing basic approximators as soon as the construction of subsequent basic approximator does not increase the accuracy). In general the bigger the value of HDAMultiMax the more accurate approximator we get. However increase of this parameter can lead to significant increase of training time or/and to overfitting in some cases.

- `HDAFDLinear`

  Values: enumeration: No, Ordinary.

  Default: Ordinary.

  Short description: include linear functions into functional dictionary used for construction of approximations.

  Description: In order to construct approximation a linear expansion in functions from special functional dictionary is used. The parameter HDAFDLinear controls whether linear functions should be included into functional dictionary used for construction of approximation or not. In general usage of linear functions as

building blocks for construction of approximation can lead to increase in accuracy, especially in the case when the approximable function has significant linear component. However usage of linear functions can lead to increase of training time as well.

- `HDAFDSigmoid`

  Values: enumeration: No, Ordinary

  Default: Ordinary.

  Short description: include sigmoid functions (with adaptation or without adaptation) into functional dictionary used for construction of approximations.

  Description: In order to construct approximation a linear expansion in functions from special functional dictionary is used. The parameter HDAFDSigmoid controls whether sigmoid-like functions should be included into functional dictionary used for construction of approximation or not. In general usage of sigmoid-like functions as building blocks for construction of approximation can lead to increase in accuracy, especially in the case when the approximable function has square-like or discontinuity regions. However usage of sigmoid-like functions can lead to significant increase of training time.

- `HDAFDGauss`

  Values: enumeration: No, Ordinary.

  Default: Ordinary.

  Short description: include Gaussian functions into functional dictionary used for construction of approximations.

  Description: In order to construct approximation a linear expansion in functions from special functional dictionary is used. The parameter HDAFDGauss controls whether Gaussian functions should be included into functional dictionary used for construction of approximation. In general usage of Gaussian functions as building blocks for construction of approximation can lead to significant increase in accuracy, especially in the case when the approximable function is bell-shaped. However usage of Gaussian functions can lead to significant increase of training time.

- `HDAInitialization`

  Values: enumeration: Deterministic, Random

  Default: Deterministic

  Short description: switch between deterministic and random initialization of parameters of the approximator.

  Description: The approximator construction technique implemented in the tool uses a random number generator for initialization of the parameters of approximator, thus the approximators constructed using the same data with different random initializations may be slightly different. If you need the approximators produced in each HDA GT run to be fully identical for the same data with other parameters

having fixed values, you must always select the value of parameter HDAInitialization equal to the value Deterministic. Otherwise (if the parameter HDAInitialization is equal to the value Random) random initialization of parameters of approximator is used. Random initialization of parameters of approximator can be used in order to obtain more accurate approximator since for different initializations the accuracies of obtained approximators may be slightly different.

- `HDARandomSeed`

  Values: integer [1, 2147483647].

  Default: 0

  Short description: Seed value for the random number generator used for random initialization of the parameters of approximator.

  Description: In case the parameter HDAInitialization is equal to the value Random then random initialization of the parameters of approximator is used. The value of the parameter HDAInitialization defines the seed value for the random number generator. There are two possibilities: the value of HDAInitialization is initialized randomly (default option) or its value is defined by the User (any positive integer).

- `HDAHessianReduction`

  Values: Real number from the interval [0, 1].

  Default: 0

  Short description: Maximum proportion of data used for evaluation of Hessian matrix.

  Description: The parameter shrinks maximum amount of data points for Hessian estimation (used in high-precision algorithm). If the parameter is equal to 0, the whole points set is used for Hessian estimation, otherwise if parameter belongs to $(0, 1]$ only part (smaller than HDAHessianReduction of whole train points) is used. Reduction is used only in case of samples bigger than 1000 input points (if number of points is smaller than 1000 points, parameter is not taken into account and Hessian is estimated by whole train sample).

  Remark: Whether GTApprox/HDAHessianReduction = 0 or not, high-precision algorithm can be nevertheless automatically disabled. This happens if

  1. $(\dim(X) + 1) \cdot p \geqslant 4000$, where $\dim(X)$ is the dimension of the input vector X and p is a total number of basis functions.
  2. $\dim(X) \geqslant 25$, where $\dim(X)$ is the dimension of the input vector X.
  3. there are no sufficient computational resources for the usage of the HP-algorithm.

4.1.4 Gaussian Processes

Short name: GP

General description: A flexible non–linear, non-parametric technique for constructing of approximation by modeling training data sample as a realization of an infinite dimensional Gaussian Distribution fully defined by a mean function and a covariance function [14, 26]. Approximation is based on a posteriori mean (for the given training data) of the considered Gaussian Process with a priori selected covariance function. AE is supported and is based on the a posteriori covariance (for the given training data) of the considered Gaussian Process. GP contains the following features:

- flexible functional class for modeling covariance function (see also description of option `GPPower` below);
- an innovative algorithm of initialization of parameters of the covariance function;
- an adaptive strategy controlling the parameters of used optimization algorithm;
- an adaptive regularization algorithm for controlling the generalization ability of the approximation;
- multi–start capabilities to globalize the optimization of the parameters;
- usage of errorbars (variances of noise in output values at training points) to efficiently work with noisy problems and improve quality of approximation and AE;
- post–processing of the results to remove redundant features in the approximation.

In short the GP algorithm works as follows:

1. Parameters of the covariance function are initialized;
2. Covariance model of the data generation process is identified by maximizing likelihood of the training data.
3. Post–processing of approximator's structure is done.

See [10] for details.

Strengths and weaknesses: GP usually demonstrates accurate behavior in the case of small and medium sample sizes. Both interpolation and approximation modes are supported (interpolation mode should not be applied to noisy problems). AE is supported. GP is designed for modeling of "stationary" (spatially homogeneous) functions, therefore GP is not well-suited for modeling spatially inhomogeneous functions, functions with discontinuities etc. GP is a resource-intensive method in terms of ram capacity, therefore large samples are not supported.

Restrictions: Large dimensions are supported. Large samples are not supported.

Options:

- `GPPower`

  Values: Real number from the interval [1, 2].

  Default: 2.

  Short description: The value of the parameter p in the p-norm which is used to measure the distance between the input vectors.

  Description: The main component of the Gaussian Process (GP) based regression is the covariance function measuring the similarity between two input points. The covariance between two input uses p-norm of the difference between coordinates of these input points. The case p = 2 corresponds to the usual Gaussian covariance function (better suited for modelling of smooth functions) and the case p = 1 corresponds to laplacian covariance function (better suited for modelling of non-smooth functions).

- `GPType`

  Values: enumeration: `Wlp`, `Mahalanobis`, `Additive`

  Default: `Wlp`

  Short description: Type of covariance function

  Description: Type of the covariance function used in the Gaussian process. Three modes are available:

  - `Wlp` refers to the widely-used exponential Gaussian covariance function.
  - `Mahalanobis` refers to squared exponential covariance function with Mahalanobis distance. In particular, the `Mahalanobis` mode can be used only with `GPPower` equal to 2.
  - `Additive` refers to additive covariance function [16,17], which is equal to the sum of products of 1-dimensional squared exponential Gaussian covariances functions. Each term in sum depends on different subsets of initial input variables. One can use option `GPInteractionCardinality` to specify the degrees of interaction for additive covariance function.

  See detailed mathematical description of used Gaussian processes types in Section A.4.3.

- `GPLinearTrend`

  Values: boolean.

  Default: off.

  Short description: Use linear trend for approximation based on Gaussian processes (GP technique only)

  Description: Allows user to take into account linear behaviour of the function being approximated. If the option is on, then covariance function of GP is a linear combination of stationary covariance (defined by the parameter `GPType`) and non-stationary covariance based on linear trend. In general, such composite covariance can lead to increase in accuracy (for functions with "linear behaviour"), but also can lead to significant increase of training time (up to three times).

- `GPMeanValue`

  Values: comma-separated list of floating point values, enclosed in square brackets. This list should be empty or the number of its elements should be equal to output dataset's dimensionality

  Default: [].

  Short description: specifies mean value of model's output mean values

  Description: models output mean values are necessary for construction of GP approximation. Model's output mean values can be defined by the User or can be estimated using the given sample (the bigger and more representative is the sample, the better is the estimate of model's output mean values). Model's output mean values misspecification leads to decrease in approximation accuracy: the larger error in output mean values leads to a worse final approximation model. If `GPMeanValue` = [], then model's output mean values are estimated using the given sample.

- `GPInteractionCardinality`

  Values: list of unique unsigned integers in the range $[1, d_{in}]$ each

  Default: [ ] (equivalent to $[1, d_{in}]$).

  Short description: defines degree of interaction between input variables used when calculating the additive covariance function.

  Description: In GP with additive kernel (`GPType` is `Additive`), the covariance function has form (A.5), i.e. it is a sum of products of one-dimensional covariance functions, where each additive component (a summand) depends on a subset of initial input variables. `GPInteractionCardinality` defines the degree of interaction between input variables by specifying allowed subset sizes, i.e. allowed values of covariance function order r in (A.5).

4.1.5 High Dimensional Approximation combined with Gaussian Processes

Short name: HDAGP

General description: HDAGP is a flexible nonlinear approximation technique, with a wide applicability in terms of space dimensions. GP usually demonstrates accurate behaviour in the case of small and medium sample sizes. However GP is mostly designed for modeling of "stationary" (spatially homogeneous) functions. HDAGP extends the ability of GP to deal with spatially inhomogeneous functions, functions with discontinuities etc. by using the HDA-based non-stationary covariance function.

Strengths and weaknesses: HDAGP usually demonstrates accurate behavior in the case of medium sample sizes. Both interpolation and approximation modes are supported (interpolation mode should not be applied to noisy problems). AE is supported. However HDAGP approximation is the slowest method compared to HDA method and GP method. HDAGP is a resource-intensive method in terms of ram capacity, therefore large samples are not supported.

Restrictions: Large dimensions are supported. Large samples are not supported.

Options: Options of HDAGP consist of both HDA options and GP options. In general HDAGP algorithm contains both HDA features and GP features. In short the HDAGP algorithm works as follows:

1. HDA approximator is constructed and base functions from this approximator are extracted;
2. Non-stationary covariance model of the data generation process is initialized by using the HDA-based non-stationary covariance function.
3. Non-stationary covariance model of the data generation process is identified by maximizing likelihood of the training data.
4. Post–processing of approximator's structure is done.

See [10] for details.

4.1.6 Sparse Gaussian Process

Short name: SGP

General description: SGP is an approximation of GP which allows to construct GP models for larger train sets than the standard GP is able to. The motivation for SGP is both to provide a higher accuracy of the approximation and enable Accuracy Evaluation for larger training sets. Current implementation of the algorithm is based on the Nyström approximation [26] and V-technique for subset of regressors method [20].

Algorithms properties: SGP provides both approximation and AE for large training set sizes (by default, SGP is used for $|S|$ from 1001 to 9999 if AE is required). All GP features, except interpolation, are supported.

In short, the algorithm of SGP works as follows:

- Choose a subset S' (base points) from the training set S. The size of S' is 1000 by default, but can be changed by user.
- Initialize parameters of the SGP covariance function as parameters of GP trained with S'.
- Calculate the posterior parameters of SGP.

Restrictions: Large training sets are supported. Large dimensions are supported. Interpolation is not supported.

Options (different from GP's, internal):

- `SGPNumberOfBasePoints`

  Values: Positive integer greater than 1.

  Default: 1000.

  Short description: Number of Base Points used to approximate full matrix of covariances between points from the training sample.

  Description: Base Points (subset of regressors) are selected randomly among points from the training sample and used for the reduced rank approximation of the full matrix of covariances between points from the training sample. Reduced rank approximation is done using Nyström method for selected subset of regressors. Note that if the value of `SGPNumberOfBasePoints` is greater than the dataset size then GP technique will be used.

- `SGPSeedValue`

  Values: integer in $[0, 2147483647]$.

  Default: 15313.

  Short description: Seed value which defines random selection of Base Points used to approximate full matrix of covariances between points from the training sample.

  Description: The value of the parameter `SGPSeedValue` defines the seed value for the random number generator. If this value is zero then random seed will be used. Seed value defines random selection of Base Points used to approximate full matrix of covariances between points from the training sample. Approximation is done using reduced rank approximation based on Nyström method for selected subset of regressors (selected Base Points).

4.1.7 Response Surface Model

Short name: RSM

General description: RSM is a kind of linear regression model with several approaches to regression coefficients estimation (see A.6). RSM can be either linear or quadratic with respect to input variables. Also RSM supports categorical input variables.

Strengths and weaknesses: A very robust and fast technique with a wide applicability in terms of the space dimensions and amount of the training data. It is, however, usually rather crude, and the approximation can hardly be significantly improved by adding new training data. In addition, the number of regression terms can increase rapidly with increasing of dimension, if quadratic RSM is used.

Restrictions: Interpolation mode is not supported, Accuracy Evaluation is not supported.

Options:

- `RSMType`

  Values: enumeration: `linear`, `interactions`, `purequadratic`, `quadratic`

  Default: `linear`

  Short description: Specifies type of the Response Surface Model

  Description: `RSMType` specifies what type of Response Surface Model is going to be constructed:

  - `linear` includes constant and linear terms;
  - `interactions` includes constant, linear, and interaction terms;
  - `quadratic` includes constant, linear, interaction, and squared terms;
  - `purequadratic` includes constant, linear, and squared terms.

- `RSMFeatureSelection`

  Values: enumeration: `LS`, `RidgeLS`, `MultipleRidgeLS`, `StepwiseFit`

  Default: `RidgeLS`

  Short description: Specifies the regularization and term selection procedures used for estimation of model coefficients.

  Description: `RSMFeatureSelection` specifies what technique is going to be used for regularization and term selection: `LS` assumes ordinary least squares (no regularization, no term selection); `RidgeLS` assumes least squares with Tikhonov regularization (no term selection); `MultipleRidgeLS` assumes multiple ridge regression that also filters not important terms; `StepwiseFit` assumes ordinary least squares regression with stepwise inclusion/exclusion for term selection.

  Remarks: Note that term here means not one of the input variables but one of the columns in the design matrix that can consist of intercept, input variables with their squares and interaction products between input variables (for mathematical details see A.6).

- `RSMMapping`

  Values: enumeration: `None`, `MapStd`, `MapMinMax`

  Default: `MapStd`

  Short description: Specifies mapping type for data preprocessing.

  Description: Specifies which technique is used for data preprocessing: `None` assumes no data preprocessing; `MapStd` assumes linear mapping of standard deviation for each variable to [-1, 1] range; `MapMinMax` assumes linear mapping of values for each variable into [-1, 1] range.

- `RSMStepwiseFit/inmodel`

  Values: enumeration: `IncludeAll`, `ExcludeAll`

  Default: `ExcludeAll`

Short description: Specifies starting model for `RSMFeatureSelection = StepwiseFit`.

Description: Specifies what terms are initially included in the model when `RSMFeatureSelection` equals `StepwiseFit`. There are two cases: `IncludeAll` assumes start with full model (all terms included); `ExcludeAll` assumes none of the terms is included in the starting step. Depending on the terms included in the initial model and the order in which terms are moved in and out, the method may build different models from the same set of potential terms.

- `RSMStepwiseFit/penter`

  Values: float in range (0., `premove`]

  Default: 0.05

  Short description: Specifies p-value of inclusion for `RSMFeatureSelection = StepwiseFit`.

  Description: Specifies maximum p-value of F-test for a term to be added into the model. The higher the value the more terms would be in general included into the final model.

- `RSMStepwiseFit/premove`

  Values: float in range [`penter`, 1.)

  Default: 0.10

  Short description: Specifies p-value of exclusion for `RSMFeatureSelection = StepwiseFit`.

  Description: Specifies minimum p-value of F-test for a term to be removed from the model. The higher the value the more terms would be in general included into the final model.

- `RSMCategoricalVariables`

  Values: list of 0-based indices of categorical variables

  Default: [ ]

  Short description: List of indices for categorical variables.

  Description: Contains indices of categorical variables in the input matrix $\mathbf{X}_{\text{train}}$.

4.2 Limitations

The common restriction on the minimum size $|S|$ of the training set for the majority of techniques is

$$|S| > 2d_{\text{in}} + 2,$$

where d_{in} is the input dimension of the data. The exceptions are RSM, LR (minimal size $|S| = 1$) and TA techniques. In case TA technique of minimal size depends on factorization (see details in Section 12) but $|S|$ should be at least 2^n where n is number of factors. Also this restriction on the minimum size $|S|$ depends on *Internal Validation* option (the above

restrictions are true for `InternalValidation = Off`). If `InternalValidation = On` then sample size $|S|$ should be higher (see details in the technical reference [15]).

As explained in Section 3.2, all conditions above refers to the *effective* values, i.e., obtained after preprocessing of the training data. An error with the corresponding error code will be returned if this condition is violated.

The maximum size of the training sample which can be processed by the tool is primarily determined by the user's hardware. Necessary hardware resources depend significantly on the specific technique. See descriptions of individual techniques.

Limitations of different approximation techniques are summarized in Table 4.1.

Technique	Compatible with			Performance on very large training sets	Other restrictions
	Lin	Int	AE		
SPLT	No	Yes	Yes		1D only
LR	Yes	No	No		
GP	No	Yes	Yes	limited by memory	
HDA	Yes	No	No	possibly long runtime	
HDAGP	No	Yes	Yes	limited by memory, possibly long runtime	
SGP	No	No	Yes		
RSM	Yes	No	No		

Table 4.1: Limitations of approximation techniques

4.3 Example: comparison of the techniques on a 1D problem

In Figure 4.1, the five techniques of **GTApprox** are compared on a 1D test problem. The parts a), b), c) of the figure represent the five approximations, their respective derivatives, and Accuracy Evaluation results (where available). AE results depend significantly on the type of approximation, see Section 9.

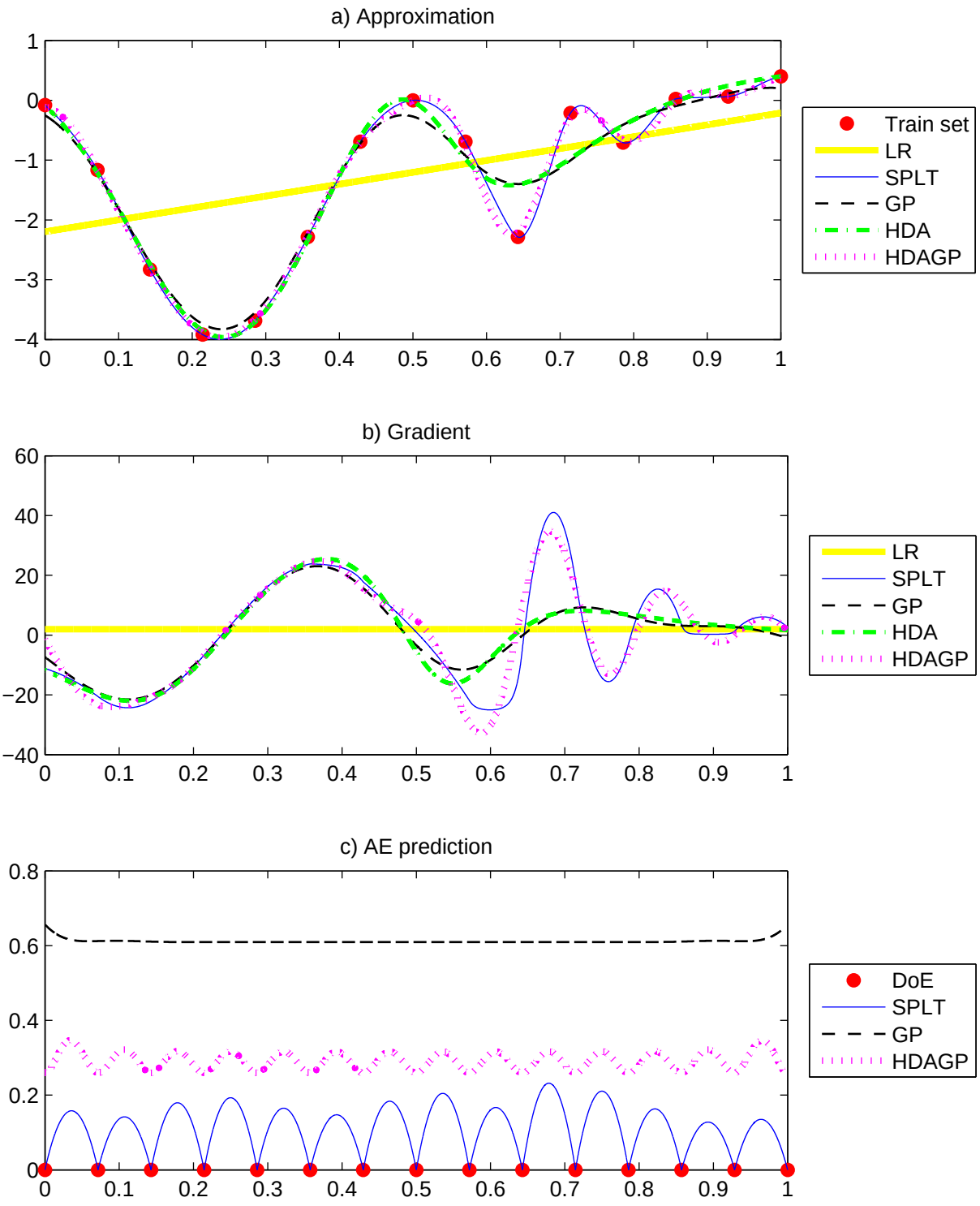


Figure 4.1: Comparison of the five approximation methods of **GTApprox** on the same 1D training data.
Note: This image was obtained using an older MACROS version. Actual results in the current version may differ.

Chapter 5

Selection of the approximation technique and the default decision tree

This section details manual and automatic selection of one of the approximation techniques described in Section 4.

5.1 Manual selection

The user may specify the approximation technique by setting the option `Technique` which may have the following values:

- `Auto` — best technique will be determined automatically (default)
- `LR` — Linear Regression
- `SPLT` — Splines with Tension
- `HDA` — High Dimensional Approximation
- `GP` — Gaussian Processes
- `HDAGP` — High Dimensional Approximation + Gaussian Processes
- `SGP` — Sparse Gaussian Processes
- `TA` — Tensor Approximation (see Chapter 12)
- `iTA` — Tensor Approximation for incomplete grids (also see Chapter 12)
- `RSM` — Response Surface Model

5.2 Automatic selection

The decision tree describing the default selection of the approximation technique is shown in Figure 5.1. The factors of choice are[1]:

[1]Some discussion of why these are essential factors of choice can be found in the development proposal [32].

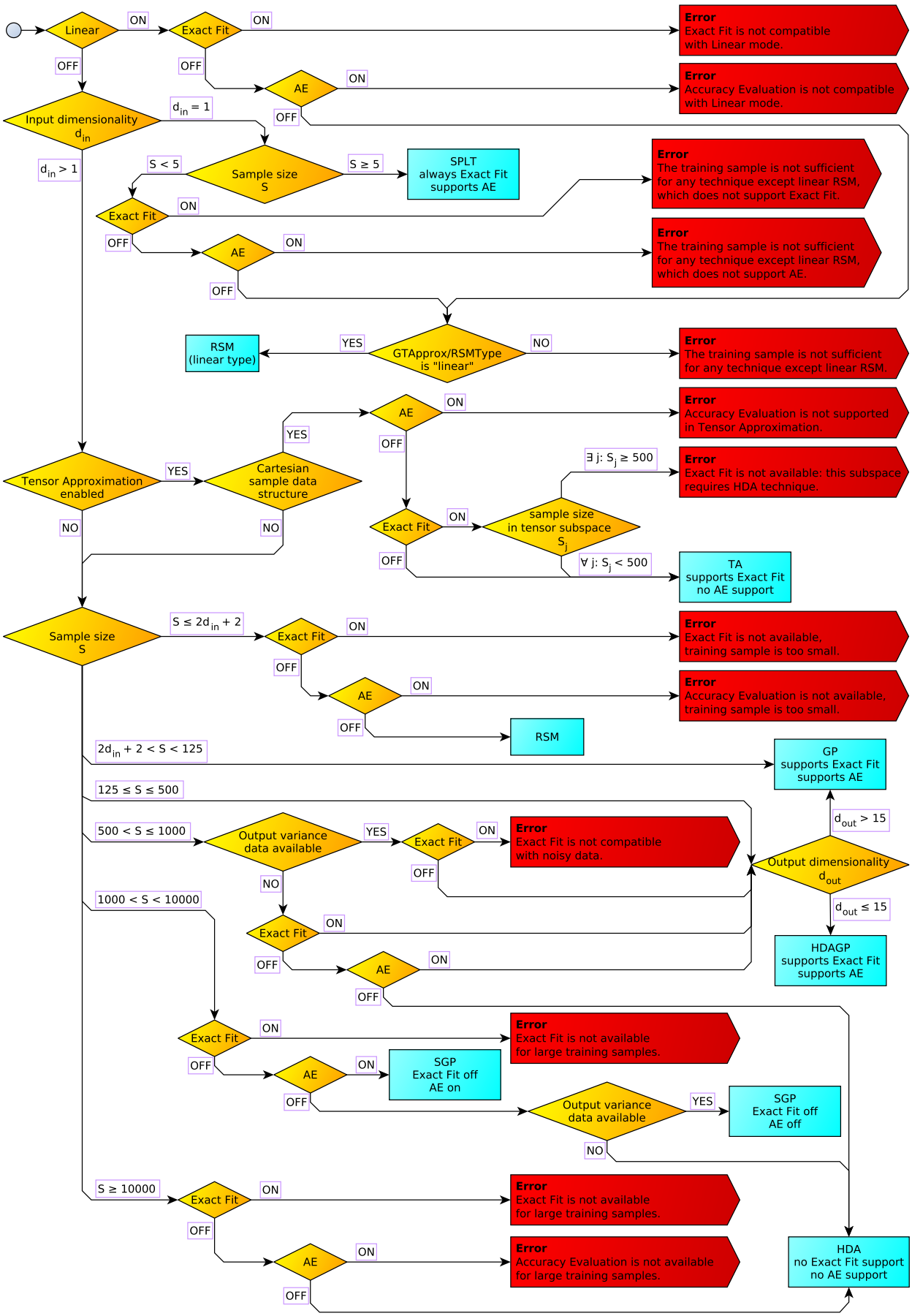


Figure 5.1: The **GTApprox** internal decision tree for the choice of default approximation methods

- Effective sample size S (the number of data points after the sample preprocessing).
- Input dimensionality d_{in} (the number of input components, or variables).
- Output dimensionality d_{out} (the number of output components, or responses).
- Linearity requirement ("Linear" node). See option `LinearityRequired`.
- Exact fit requirement ("Exact Fit" nodes). See option `ExactFitRequired`.
- Accuracy evaluation requirement ("AE" nodes). See option `AccuracyEvaluation`.
- The availability of output variance data (responses with "errorbars"). See Section 6.6 for details.
- Tensor approximation switch ("Tensor Approximation enabled" node). See option `EnableTensorFeature`. Note that the Tensor Approximation technique requires a specific sample structure and has several more options not shown on Figure 5.1, which may affect the automatic selection. See Chapter 12 for more details, in particular Section 12.3 for the full TA decision tree.

The result is the constructed approximation, possibly with an accuracy prediction, or an error. The selection is performed in agreement with properties of individual approximation techniques as described in Section 4. In particular:

- For 1D case, the default technique is SPLT except when the sample is very small: if sample size is 4 points or less, only a linear model is possible. In all other 1D cases, SPLT has a good all–around performance. It is very fast and can efficiently handle both small (starting with 5 points) and very large training sets. Also, SPLT is inherently an exact-fitting technique, and therefore it should be manually replaced with a smoothing technique if the training data is very noisy; see Section 6.4.
- In dimensions 2 and higher, the default techniques are GP, HDAGP and HD, depending on the size $|S|$ of the training set:
  - If $|S| \leqslant 2d_{\text{in}} + 2$, then the default technique is RSM, which is robust for such small sets.
  - If $2d_{\text{in}} + 2 < |S| < 125$, then the default technique is GP, which is normally efficient for rather small sets.
  - If $125 \leqslant |S| \leqslant 500$, then the default technique is HDAGP, combining GP with HDA.
  - If $500 < |S| \leqslant 1000$, then the technique depends on whether Accuracy Evaluation or Interpolation is required:
    - If both **AE** and **Int** are `off`, then the default technique is HDA;
    - Otherwise, the default technique is HDAGP.
  - If $1000 < |S| < 10000$, then the technique depends on whether Accuracy Evaluation is required:
    - If **AE** is `off`, then the default technique is HDA;
    - Otherwise, the default technique is SGP.

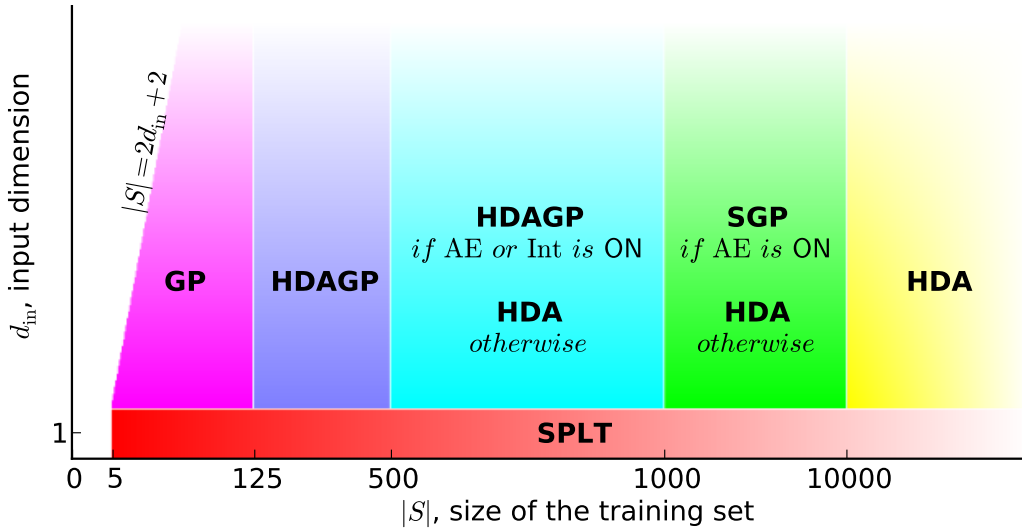


Figure 5.2: The "sample size vs. dimension" diagram of default techniques in **GTApprox**

– If $|S| \geqslant 10000$, then the default technique is HDA, which can handle the largest sets (note, however, that exact fitting mode and accuracy evaluation are not available for it).

The threshold values $2d_{\text{in}} + 2$, 125, 500, 1000 and 10000 for $|S|$ have been set based on previous experience and extensive testing.

In Figure 5.2 the "sample size vs. dimension" diagram for the default choice is shown.

Chapter 6

Special approximation modes

In some cases, it is desirable to adjust the type or character of the approximation according to the prior domain–specific knowledge of the problem or specific requirements to the approximation. **GTApprox** supports several special approximation modes which can be useful in some applications. The important special mode of tensor product of approximations for factored data is, due to its complexity, covered in a separate chapter (see Chapter 12).

6.1 Linear mode

The linear mode can be set by the option `LinearityRequired` (default value is `off`). If this option is turned on, then the approximation is constructed as a linear function which optimally fits the training data. If the switch is off, then no linearity-related condition on the approximation is imposed: it can be either linear or non-linear depending on which fits the training data best. See Figure 6.1.

By default, the linear mode is implemented by the linear regression of the training data, see the LR algorithm in Section 4. An alternative way to turn the linear mode on is to directly choose this algorithm, see Section 5.

For a number of reasons, linear approximations are often useful in applications:

- A general nonlinear approximation constructed by **GTApprox** is usually hard-to-describe; the user might be interested in some abstract properties of the approximation which are hard to determine even though for any specific input the evaluation of the approximation is very fast. In contrast, a linear approximation has a very simple and transparent form.

- Linear regression is a crude approximation method, but its advantage is a high universality and robustness. It is applicable in a large range of dimensions and training sample sizes, and insensitive to the noise.

- A linear approximation may be sufficient for some data analysis task, e.g. a crude assessment of the response function's global sensitivity with respect to the independent variables can be achieved by computing the coefficients of the linear approximation.

- Linear regression is normally the method of choice if the size of the training sample is somewhat greater but comparable to the input dimension of the problem.

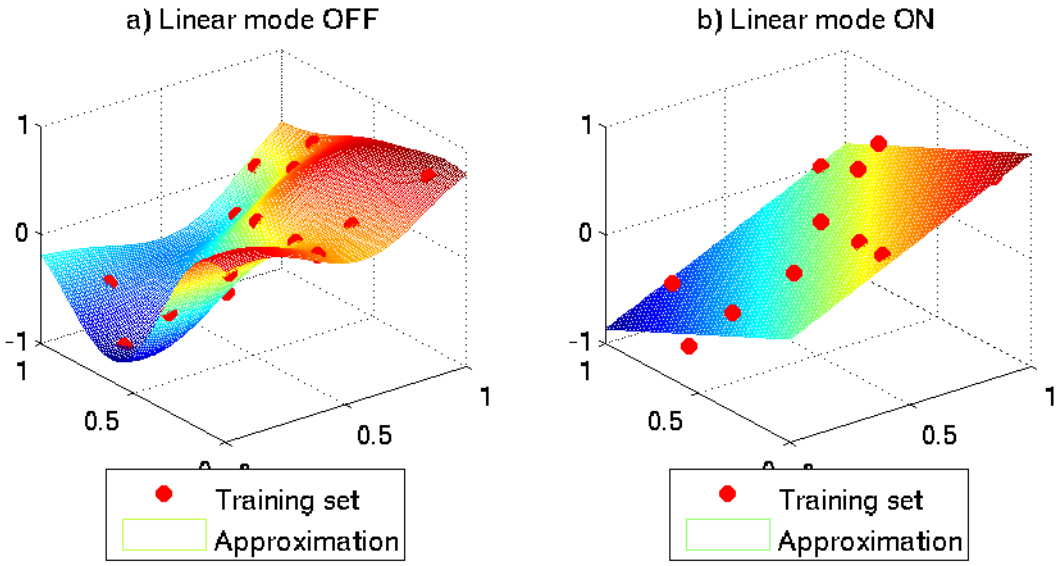


Figure 6.1: Examples of approximations constructed from the same training data with linear mode OFF or ON.

Linear mode is incompatible with the interpolation mode (see Section 6.2), since a strict interpolation by a linear function is only possible in exceptional or degenerate cases. An error will be returned if both of these options are switched on.

Also, linear mode is not supposed to be compatible with Accuracy Evaluation (see Section 9), since the two methods are different in the intended applications: the former is suitable for a crude but robust approximation with little training data or in the presence of noise, while the latter is intended for a detailed analysis of noiseless models with sufficiently rich training data.

If both the linear mode and one of the specific approximation algorithms are selected by the user as explained in Section 5, then an error will be raised if the selected algorithm is GP, HDAGP, SGP or SPLT. However, the linear mode is compatible with HDA, which can produce a linear approximation if required.

6.2 Interpolation mode

The interpolating mode can be set by the option `InterpolationRequired` (default value is `off`). If this switch is turned on, then the constructed approximation will go through the points of the training sample (see Figure 6.2). If the switch is off, then no interpolation condition is imposed, and the approximation can be either interpolating or non-interpolating depending on which fits the training data best.

The interpolating mode is computationally demanding and therefore restricted to moderately sized training samples. Of the specific approximation techniques described in Section 5, this mode is supported by GP, HDAGP and SPLT (support for the latter is trivial as SPLT is always interpolating in **GTApprox**). It is not supported by HDA, SGP and LR.

The following guidelines are important in choosing whether to switch the strict interpolation mode on or off:

- If the approximation has a low error on the training sample (in particular, if it is a strict

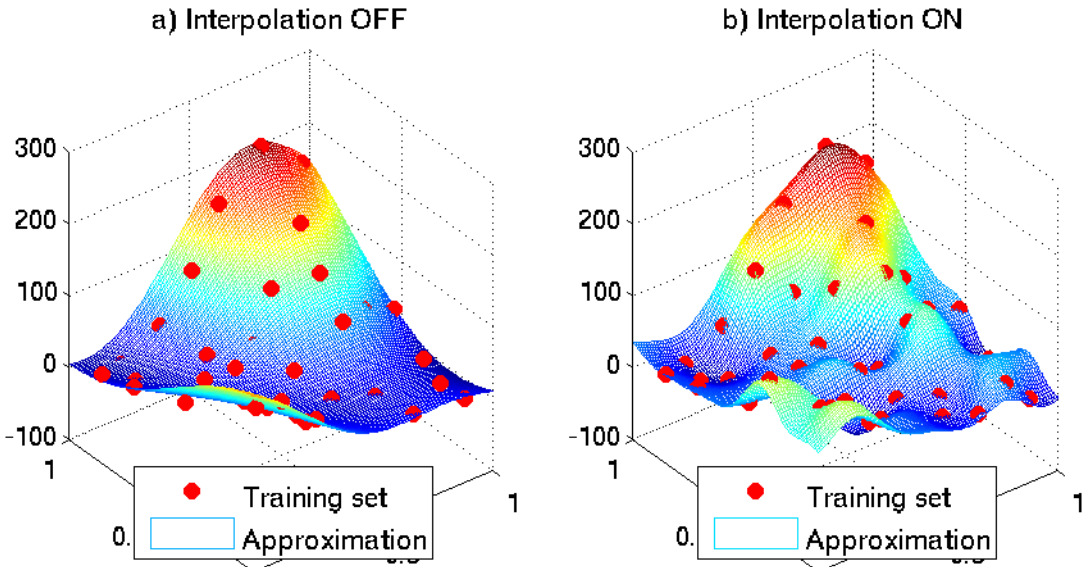


Figure 6.2: Examples of approximations constructed from the same training data with Interpolation OFF or ON.

interpolation), it does not mean that this approximation will be just as accurate outside of the training sample. Very often (though not always), requiring an excessively small error on the training sample leads to an excessively complex approximation with low predictive power – a phenomenon known as *overfitting* or *overtraining* (see, e.g., [18,31] and Section 7). If the strict interpolation mode is off, **GTApprox** attempts to avoid overfitting.

- The interpolation mode is inappropriate for noisy models. Also, if the training sample is highly irregular or the approximated function is known to be singular, then the interpolation mode is not recommended, as in this case the approximation tends to be numerically unstable. On the whole, strictly interpolating approximations are more flexible but less robust than non-interpolating ones.

- The interpolation mode may be useful, e.g., if the default approximation (with the interpolation mode off) appears to be too crude. In this case, turning the interpolation mode on may (but is not guaranteed to; the opposite effect is also possible) increase the accuracy.

Because of numerical limitations and round-off errors, minor discrepancies can be observed in some cases between the training sets and the interpolating approximations constructed by **GTApprox**. These discrepancies typically have relative values $\lesssim 10^{-5}$ and are considered negligible.

6.3 Componentwise training of the approximation

In case of the multidimensional output of the response function ($d_{\text{out}} > 1$), there are, in general, two modes to construct the approximation:

1. jointly for all scalar components of the output;

2. separately for each scalar component of the output.

The choice of the mode is regulated by the option `Componentwise` . The default value of this option is `off`, i.e. the first mode is used. On the whole, the differences between the two modes are:

- the joint mode is faster;
- the joint mode attempts to take into account the possible dependency between different components of the output (e.g., this is so if different components of the output describe values of a smooth process at different time instances or describe different points on family of curves or surfaces depending on some parameters);
- the total approximation obtained with the joint mode may have (but not always does) a lower overall accuracy than the set of separately obtained approximations for each component.

In some cases, however, changing the mode may have no or very little effect on the approximation.

6.4 Noisy problems

By *noisy* problems one usually understands those problems where the response function is a regular function perturbed by a random, unpredictable noise. In these problems one is typically interested in some noise–filtering and smoothing of the training data so as to recover the regular dependence rather than reproduce the noise; in particular, strict interpolation is unnecessary and possibly undesirable. The extreme case of noisy problems are those with training data containing points with the same input, but different output (e.g., random variations in the results of different experiments performed under the same conditions). The sensitivity of an approximation method to the noise is usually positively correlated with the interpolating capabilities of the method and negatively correlated with its crudeness.

SPLT	Strictly interpolating, very noise–sensitive.
GP, HDAGP, SGP	Not strictly interpolating in general, tend to somewhat smoothen the data (degree of smoothing depends on a particular problem).
HDA	Mostly smoothing. Sensitivity can be further tuned by adjusting the complexity parameters of the approximation through advanced options, see Section 4.1.3. Can be applied to very noisy problems with multiple outputs corresponding to the same input.
LR	The crudest and most robust method. Very noise–insensitive.

Table 6.1: Noise–sensitivity of different approximation techniques with default settings in **GTApprox**

Table 6.1 summarizes how the methods of **GTApprox** can be ordered with respect to their noise sensitivity, from the most sensitive to the least sensitive. The table refers to the default versions of the methods, with the interpolation mode turned off (see Section 6.2). User is advised to adjust the level of smoothing in particular problems by choosing the appropriate approximation technique of **GTApprox**.

An example of the application of different methods of **GTApprox** to a 1D noisy problem is shown in Figure 6.3.

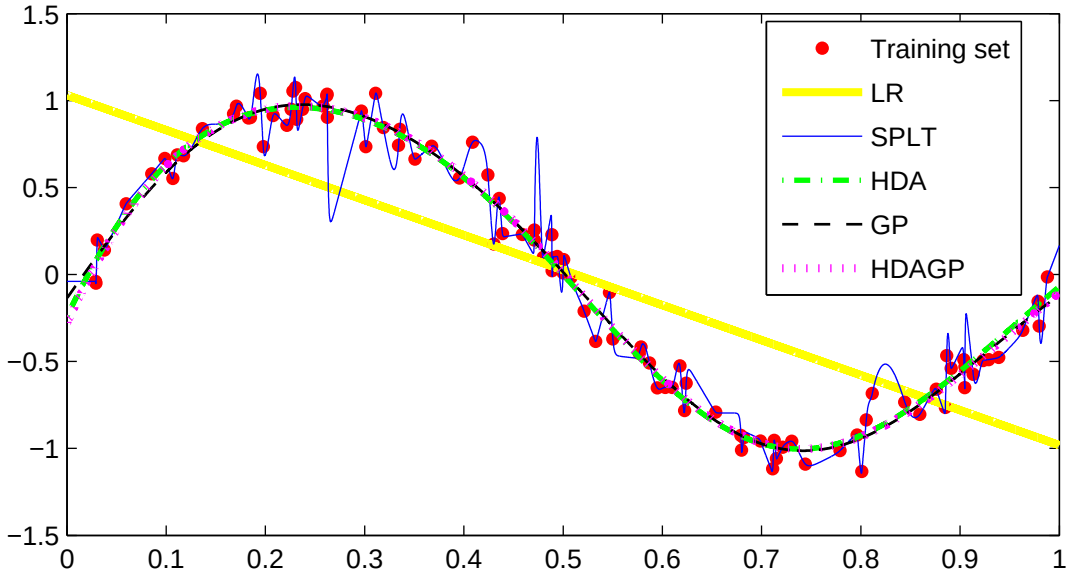


Figure 6.3: Different approximation techniques on a noisy problem. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

6.5 Heteroscedastic data

It is usually assumed that the noise in output data sample has uniform properties and can be modelled by independent identically distributed multivariate normal variables (particular case of homoscedastic noise). However, in some approximation problems noise can be heteroscedastic, e.g. when output variance depends on the point location in the design space. For such problems, the noise in some regions may be negligible, while in other regions the presence of noise has to be taken into account to build an accurate enough approximation and provide consistent accuracy evaluation.

To resolve such issues, **GTApprox** implements a dedicated algorithm based on Gaussian processes. It assumes the following form of the output:

$$f(X) = f_1(X) + \varepsilon(X),$$

where $f_1(X)$ is a realization of a Gaussian process, $\varepsilon(X)$ is white noise with zero mean and variance given as $\exp(f_2(X))$ where $f_2(X)$ is also a realization of a Gaussian process. During training we estimate parameters of Gaussian processes $f_1(X)$ and $f_2(X)$; $f_1(X)$ models the true function, while $f_2(X)$ models the noise in the output data.

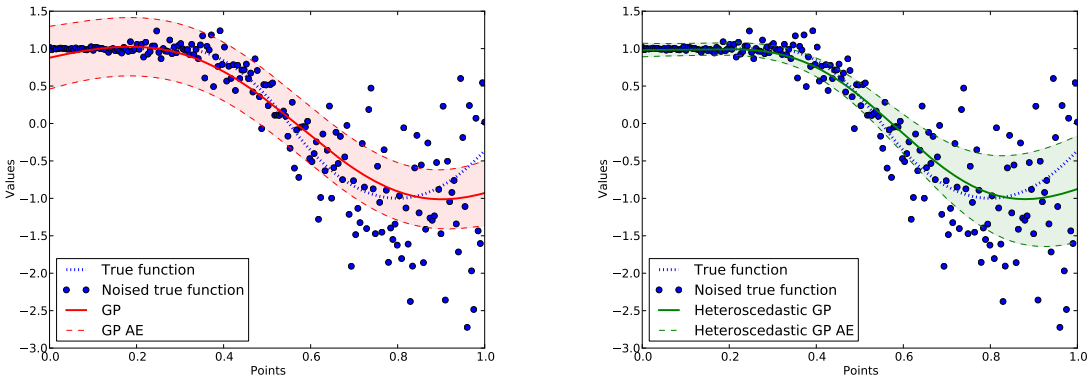


Figure 6.4: The results of GP technique, ordinary vs. heteroscedastic version

The above algorithm is used only when the `Heteroscedastic` option is on. By default it is assumed that the data is homoscedastic, and heteroscedastic version of Gaussian process is not used.

Specific features of the heteroscedastic Gaussian processes are:

- This algorithm is only available when `GP` or `HDAGP` technique is used.
- Accuracy evaluation is available and typically is significantly more accurate than that of `GP` and `HDAGP` with `Heteroscedastic` option off, if the input data is really heteroscedastic.
- Due to the limitations of `GP` and `HDAGP` techniques, their heteroscedastic versions are only useful if the sample size S does not exceed a few thousands of points.
- This algorithm is not compatible with the Mahalanobis covariance function for GP and with explicit specification of the input variance (see section 6.6).

An example result of using the heteroscedastic algorithm is shown on figure 6.4. There, original function is one-dimensional piecewise quadratic function. It was corrupted with white noise which quadratically increases its standard deviation with larger values of input X. Note how accuracy evaluation in the second case (heteroscedastic GP, right) correctly reflects the heteroscedasticity in the given data sample, compared to ordinary GP (left).

6.6 Data with errorbars

If the user has specific knowledge about the level of noise in his data, namely variances of noise in output values at train points, these values (errorbars) can be set as another input of GTApprox. If errorbars are set then algorithms of GTApprox will use them for additional smoothing of the model. This functionality can be used only with techniques based on Gaussian Processes: GP, HDAGP and SGP. Depending on the type of covariance function used in GP model, usage of errorbars will have different influence on approximation. The best effect is expected for `Wlp` covariance function with `GPPower = 2` (See Section 4.1.4 for details of GP options). For `1 <= GPower < 2` the improvement of the model is less significant. Errorbars are especially helpful if level of noise is relatively high (5 or more percents of output function values). The mathematical description is available in the Section A.4.5.

6.7 Smoothing

GTApprox has the option of additional smoothing the approximation *after* it has been constructed. The user can transform the trained model into another, smoother model. Smoothing affects the gradient of the function as well as the function itself. Therefore, smoothing may be useful in tasks for which smooth derivatives are important, e.g., in surrogate-based optimization.

Details of smoothing method depend on the technique that was used to construct the approximation:

- SPLT. Implementation of smoothing in **GTApprox** is based on article [8] and consists in convolving the approximation with a smoothing kernel K_h, where h is the kernel width. The kernel width controls the power of smoothness: the larger the width, the more intensive smoothing we get.
- GP, SGP, HDAGP. Smoothing is based on adding a linear trend to the model which leads to new covariance function. Degree of smoothness depends on the coefficient of non-stationary covariance term (based on linear trend): the larger this coefficient, the closer model to the linear one.
- HDA. Smoothing is based on penalization of second order derivatives of the model. HDA model represents as linear decomposition of nonlinear parametric functions so Decomposition coefficients are re-estimated (via penalized least squares) during smoothing. Smoothness is controlled by regularization parameter (the coefficient at penalty term of least square problem).
- LR. Linear approximations are not affected by smoothing

User can assign two main parameters of smoothing procedure which control the following things:

- How to smooth the model (isotropic or anisotropic smoothing). Anisotropic means different degree of model smoothness along different directions. Anisotropic smoothing isn't possible for GP-based techniques (GP, HDAGP, SGP).
- How to determine required smoothness level. There are two possibility:
  - Use abstract parameter smoothness from the interval $[0, 1]$. smoothness $= 0$ corresponds to unsmoothed model(as it was trained), smoothness $= 1$ corresponds to extremely smooth model (almost linear). This smoothing parameter has no physical meaning, it is only a relative measure of smoothness.
  - Provide a sample and specify some type of error and corresponding threshold. Algorithm finds the smoothest model which ensures required error level (less than threshold) on the given sample.

Also smoothing algorithm allows to introduce different smoothing parameters for different components of multidimensional output. See Technical References for the implementation details.

In Figures 6.5 and 6.6 we show examples of smoothing in 1D and 2D.

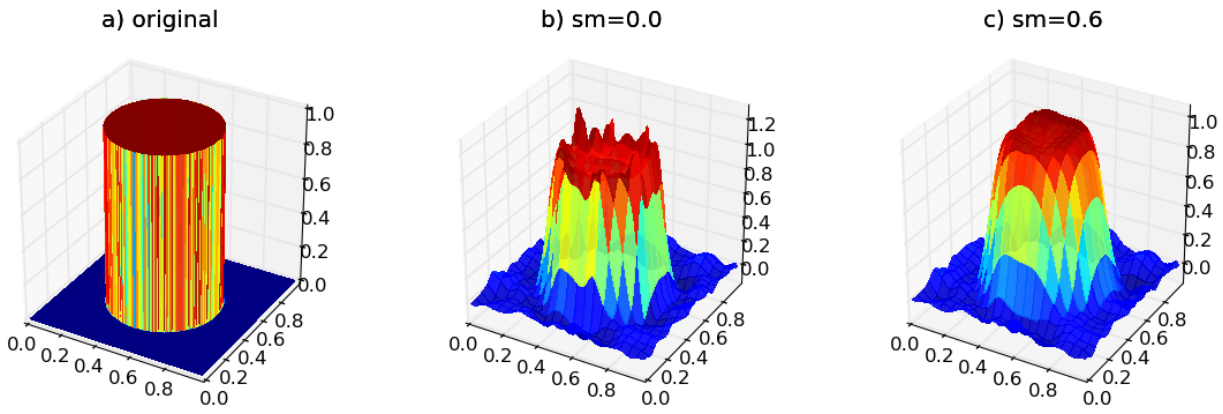


Figure 6.5: 2D smoothing: a) original function; b) non-smoothed approximation; c) smoothed approximation

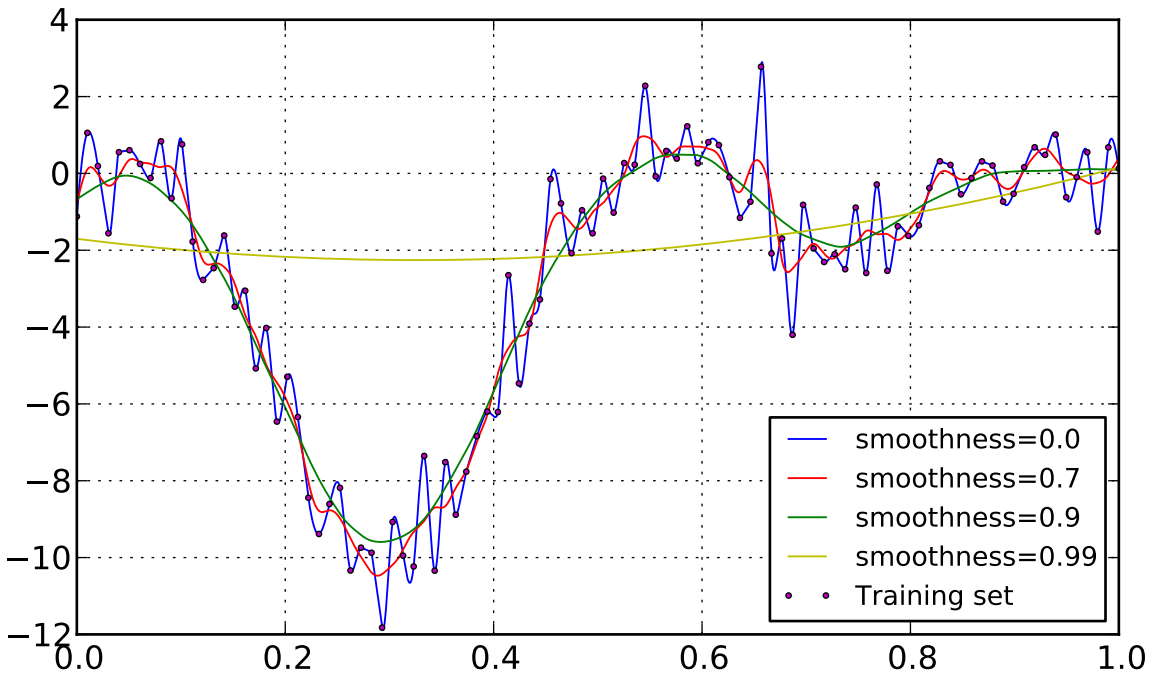


Figure 6.6: 1D smoothing with different values of the parameter smoothness. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

Chapter 7

Control over accuracy and validation of the model

7.1 Overview of accuracy estimation and related functionality of the tool

Suppose that a training set (X_k, Y_k) represents an unknown response function $Y = f(X)$, and $\widehat{f}$ is an approximation of f constructed from this training data. An important property of the approximation is its *predictive power* which is understood as the agreement between f and $\widehat{f}$ on points X *not* belonging to the training set. A closely related concept is *accuracy* of the approximation which is understood as the deviation $|\widehat{f} - f|$ on the design space of interest. Accuracy is often measured statistically, e.g., standard accuracy measures are the *Mean Absolute Error (MAE)* and *root-mean-squared error (RMS)*:

$$\mathrm{MAE} = \frac{1}{|D|}\int_D |\widehat{f} - f|,$$
$$\mathrm{RMS} = \Big(\frac{1}{|D|}\int_D |\widehat{f} - f|^2\Big)^{1/2},$$

where $|D|$ is the volume of the design space. Note that the above definitions essentially depend on the design space on which the approximation is considered.

In practice, since the response function is unknown, the accuracy is usually estimated by invoking a *test set* $(X'_n, Y'_n = f(X'_n))_{n=1}^N$ and replacing the above expressions with

$$\mathrm{MAE} \approx \frac{1}{N}\sum_{n=1}^{N} |\widehat{f}(X'_n) - f(X'_n)|,$$
$$\mathrm{RMS} \approx \Big(\frac{1}{N}\sum_{n=1}^{N} |\widehat{f}(X'_n) - f(X'_n)|^2\Big)^{1/2}.$$

The test DoE $\{X'_n\}$ should be:

- possibly large and uniformly distributed in the design space;
- possibly independent of the training DoE.

The first requirement is needed to ensure the closeness of the error estimates to the actual values. The second requirement ensures approximation's efficiency with respect to its

predictive capabilities rather than reproducing the already known information, and ensures against the phenomenon known as *overfitting* or *overtraining* (see, e.g., [18, 31]).

This phenomenon consists in getting the approximation to be very accurate on the training DoE, at the cost of excessively increasing approximation's complexity which leads to a less robust behavior and ultimately lower accuracy on points not belonging to the training set (see, e.g., [29] for the classical example of Runge's phenomenon). The errors of an overfitted approximation on a training set are usually much lower that the errors on the independent test set.

Interpolating property of the approximation is often connected with overfitting. Though for very accurate training data and simple response functions interpolation may be acceptable and even desirable (see Figure 2.1), for less regular or noisy functions interpolation is associated with overfitting and is to be avoided (see Figures 6.2 and 6.3). Modern approximation techniques usually contain some internal mechanisms preventing the approximation from overfitting; in particular, the non–linear techniques of **GTApprox** have such mechanisms.

In practice, in the absence of other information about the approximated function, testing the approximation on a properly selected test set is the most reliable way of measuring approximation's accuracy. This, however, requires the user to have a sufficiently large test set independent of the training set. Typically, knowledge of the response function f is limited, and one has to divide the available data into a training and a testing part. The allocation of only a part of data, compared to the whole amount, to the training set necessarily reduces, in general, approximation's accuracy, but allows one to estimate it, which would otherwise not be possible.

GTApprox offers an error estimating functionality, *Internal Validation (IV)*, based on a closely related idea of cross-validation (see, e.g. [1, 21]). When constructing the approximation, the particular algorithm used for that is applied to various subsets of the training set, and is tested on the complementary subsets. In this way, the user gets an approximation constructed using the whole amount of training data as well as an estimate of its accuracy, while being relieved of the need to preprocess the data into a training and a testing parts or perform any other operations with the data. See Section 10 for more details.

In addition to Internal Validation, **GTApprox** features another error analysis procedure of a different nature, *Accuracy Evaluation (AE)*, see Section 9. The purpose of this procedure is to estimate approximation's error as a function of the input vector $X \in D$, rather than produce an averaged error measure as in IV.

The differences between IV and AE can be explained as follows:

- AE provides a more informative output (a function on the design space) than IV (a table of estimated averaged errors).
- More informative output of AE comes at the cost of a somewhat reduced applicability range. While IV is practically universally applicable, AE is restricted to moderately sized training sets. See Section 9 for more details.
- AE estimates the error of the *constructed approximation*, while IV rather estimates the efficiency of the *approximating algorithm* on the training data, by analyzing errors of other approximations constructed with this algorithm and with these data.
- As with any prediction, error estimates of AE are based on some preliminary internal assumptions about the model which may be more or less appropriate for the problem

in question. In contrast, IV makes no such assumptions and consists of only straightforward statistical operations.

- On the whole, AE is appropriate if the relative values of errors at different design vectors are of interest, while IV is more suitable if the overall approximation's accuracy is of interest.

- IV is time–consuming, as it involves a multiple training of approximations. In contrast, AE does not require much time beyond that required to construct the main approximation.

7.2 Selection of the training data

GTApprox has been designed to operate with almost any training set in any input dimension, subject only to the minimum size restriction (see Section 4.2) and hardware limitations. Nevertheless, the choice of the training data, in particular DoE, is important for the performance of the tool.

As a general rule, a larger training set typically (though not always) will allow to construct a more accurate approximation (possibly at the cost of a longer training time).

Very often, training data are expensive to obtain, but the user has the freedom of forming a training set of a fixed size by choosing a DoE on which the response function is to be evaluated. In such cases, it is normally preferable to use a DoE which is in some sense reasonably uniformly distributed in the design space. Various types of these DoE are known in the literature, such as full–factorial design, (optimal) latin hypercubes, Sobol sequences, etc., see, e.g., [30]. A number of such DoE are provided by the Generic Tool for DoE (**GT DoE**). Some of the DoE, such as Sobol sequences, can be naturally expanded preserving uniformity if additional resources for response function evaluation become available (also, see Section 9.2.2 for another approach to this task).

In Figure 7.1, some examples of different DoE are shown. The DoE a) is reasonably uniform. The DoE b) is degenerate as it is a subset of a 1D subspace in a 2D design space. The approximation constructed using this DoE will likely lack accuracy outside this 1D subspace. The DoE c) is uniform, but only in the lower left quadrant of the design space; the approximation constructed using this DoE will likely lack accuracy outside this quadrant. The DoE d) has points placed very close to each other; this is likely to cause additional numerical difficulties for the approximation algorithm and thus reduce approximation's accuracy, especially if the response function is noisy (see Section 6.4).

A popular choice of DoE is random (Monte–Carlo) sampling, due to its universality and simplicity of implementation. Note, however, that in low dimensions random points tend to crowd, and a random DoE is typically less uniform than the types mentioned above, see Figure 7.2. A random DoE is thus not the best choice for noise–sensitive approximation methods in low dimensions.

7.3 Limitations of accuracy estimation

Though accuracy estimation functionality provided by **GTApprox** in the form of IV and AE can be useful in many cases, as demonstrated, e.g., by examples in Sections 9 and 10, it should be remembered that accuracy of the approximation and the error predictions

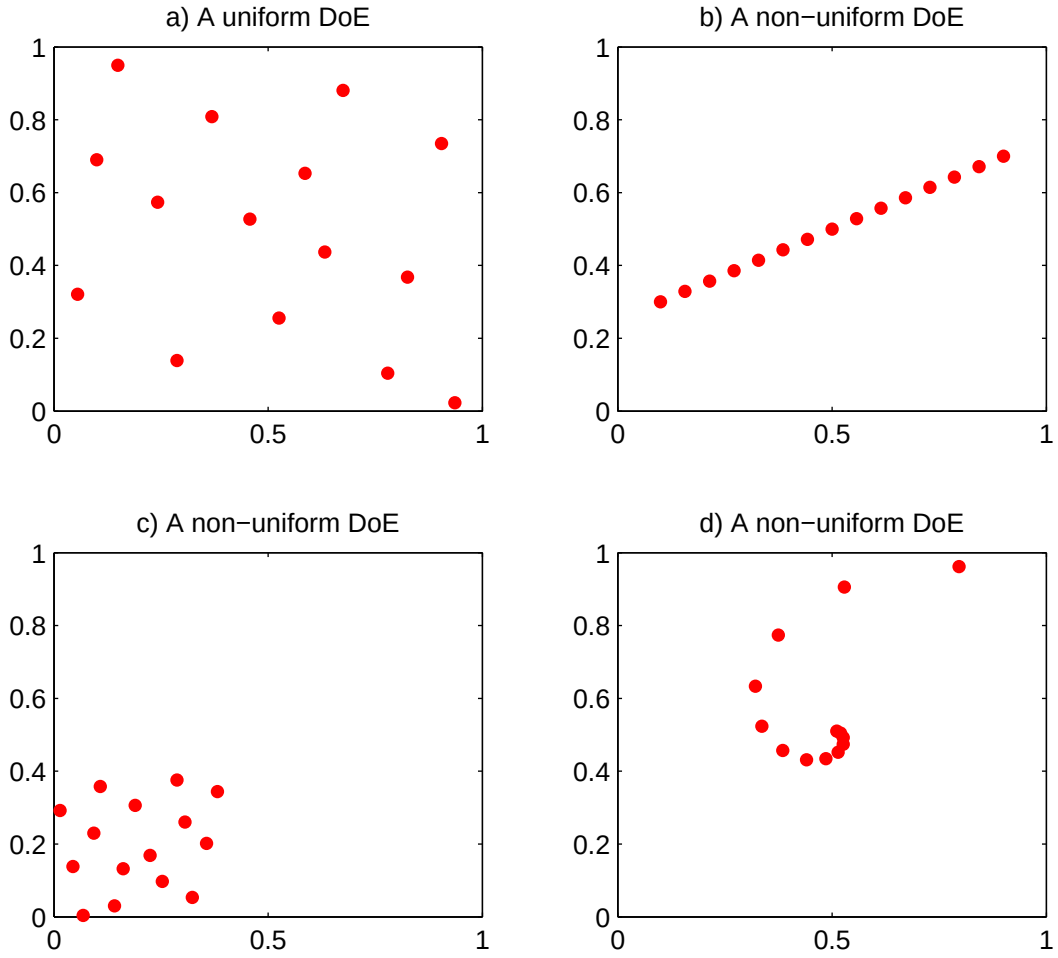


Figure 7.1: Examples of uniform and non-uniform DoE in the square $[0,1]^2$

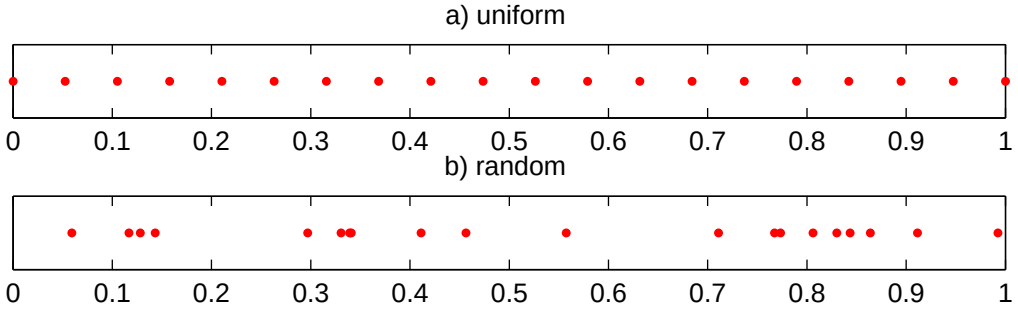


Figure 7.2: A uniform and a random DoE of the same size in 1D

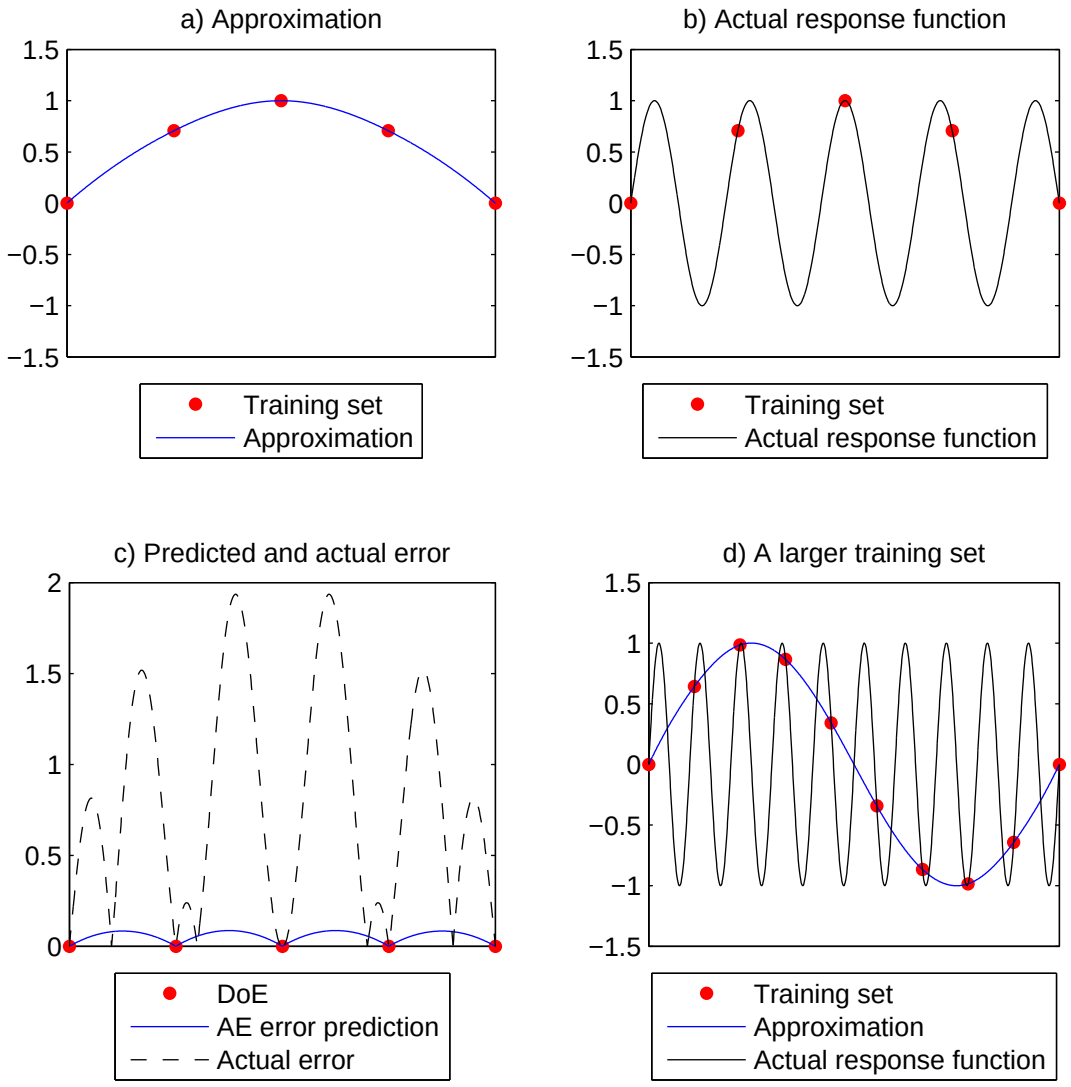


Figure 7.3: An example of a misleading training set. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

is fundamentally limited by the information contained in the training set and the natural interpretation of this information.

This can be illustrated with the following simple example. Suppose that given is the training set of 5 points on a strictly uniform DoE in the segment $[0, 1]$, as shown in Figure 7.3a). **GTApprox** constructs an approximation with the default parameters, also shown in Figure 7.3a). The approximation looks quite natural.

It may well happen, however, that the actual response function has a much more complicated form, as shown in Figure b). Note that nothing in the provided training set of 5 points suggests this complicated behavior.

In Figure c), the actual and the AE predicted errors are shown. The low AE prediction for the error appears reasonable given the simple form of the trend observed in the training data. However, the actual errors turn out to be much greater. An application of IV in this problem would also estimate the approximation error as very low, since the training data do not hint at the possibility of the strong deviation of the actual response function from the

approximation.

The above difficulty is of course possible for any size of the training set, see, e.g., Figure 7.3d).

This artificial example shows why it is not possible, in general, to accurately predict approximation's error, especially if the training data are scarce. The only way to reveal the strong oscillations of the actual response function would be an additional sampling resolving the high–frequency modes. In general, neither approximation nor AE prediction can be efficient if the information contained in the training data is significantly inadequate to the complexity of the response function. The accuracy estimates of **GTApprox** offer reasonable suggestions consistent with the user–provided training data, but their full agreement with the actual errors cannot be guaranteed.

Chapter 8

Control over training time

8.1 General notes

As **GTApprox** normally constructs approximations by complex nonlinear techniques, this process may take a while. In general, the quality of the model trained by **GTApprox** is positively correlated with the time spent to train it. **GTApprox** contains a number of parameters affecting this time. The default values of the parameters are selected so as to reasonably balance the training time with the accuracy of the model, but in some cases it may be desirable to adjust them so as to decrease the training time at the cost of decreasing the accuracy, or increase the accuracy at the cost of increasing the training time. The following general recommendations apply:

- The fastest approximation algorithms of **GTApprox** are, by far, SPLT and LR (see Section 4). They have, however, a limited applicability: SPLT is exclusively for 1D, and LR is too crude in many cases.
- Internal Validation involves a multiple training of the approximation which slows down the training time by a factor of $N_{\text{tr}} + 1$, where N_{tr} is the number of IV training/validation sessions. To speed up, turn the Internal Validation off or decrease N_{tr}. See Section 10 for details.
- The only nonlinear approximation algorithm effectively available in **GTApprox** for very large training sets (larger than 10000) in dimensions higher than 1 is HDA. This algorithm can be quite slow on very large training sets, but it has several options which can be adjusted to decrease the training time. See Section 4.1.3 for details.

In addition, **GTApprox** has a special option `Accelerator` which allows the user to tune the training time by simply choosing the level 1 to 5; the detailed specific options of the approximation techniques are then set to pre-determined values. See Section 8.2.

We emphasize that the above remarks refer to the *training time* of the model, which should not be confused with the *evaluation time* (recall the two different phases shown in Figure 2.2). After the model has been constructed, evaluating it is usually a very fast operation, and this time is negligible in most cases (though very complex models, in particular tensor product of approximations, see Section 12).

8.2 Accelerator

The switch `Accelerator` allows the user to reduce training time at the cost of some loss of accuracy. The switch takes values 1 to 5. Increasing the value reduces the training time and, in general, lowers the accuracy. The default value is 1.

Increasing the value of `Accelerator` simplifies the training process and, accordingly, always reduces the training time. However, the accuracy, though degrades in general, may change in a non-monotone fashion. In some cases, the accuracy may remain constant or even improve.

`Accelerator` acts differently depending on the approximation technique.

- **HDA:** Here, the following rules apply:

  1. `Accelerator` changes the default settings of HDA.
  2. User-defined settings of HDA have priority over those imposed by `Accelerator`.
  3. The technique's parameters are selected differently for very large training sets (more than 10000 points) and moderate sets (less than 10000 points).

  In brief, the following settings are modified by `Accelerator` (see Section 4.1.3):

  - `HDAPhaseCount` is decreased;
  - `HDAMultiMax` is decreased;
  - Usage of Gaussian functions is restricted, and with large training sets they are not used at all (`HDAFDGauss`);
  - `HDAHPMode` is off for large training sets.

- **GP:** `Accelerator` speeds up training by adjusting internal settings of the algorithm.
- **HDAGP:** As HDAGP is a fusion of HDA and GP, `Accelerator` in this case adjusts settings of the latter two techniques as described above.
- **SGP:** As SGP is based on GP, `Accelerator` in this case adjusts settings of GP as described above.
- **LR, SPLT:** Training time of these techniques is negligible and there is no need to improve it. Accordingly, `Accelerator` does not affect these techniques.

Examples To illustrate the effect of `Accelerator`, we show how it changes the training time and accuracy of the default approximation on a few test functions. We consider the following functions:

- **Rosenbrock:**

$$Y = \sum_{i=1}^{d-1} (1 - x_i)^2 + 100 \left(x_{i+1} - x_i^2\right)^2, \quad X = (x_1, \ldots, x_d) \in [-2.048, 2.048]^d$$

- **Michalewicz:**

$$Y = \sum_{i=1}^{d} \sin(x_i) \sin\left(\frac{x_i^2}{\pi}\right), \quad X \in [0, \pi]^d$$

- **Ellipsoidal:**

$$Y = \sum_{i=1}^{d} i x_i^2, \quad X \in [-6, 6]^d$$

- **Whitley:**

$$Y = \sum_{i=1}^{d} \sum_{j=1}^{d} \Big(c + \frac{c}{b} \Big(a \left(x_i^2 - x_j \right)^2 + (c - x_j)^2 \Big)^2 - \cos \Big(a \left(x_i^2 - x_j \right)^2 + (c - x_j)^2 \Big) \Big),$$
$$a = 100,\ b = 400,\ c = 1, \quad X \in [-2, 2]^d$$

We consider these functions for different input dimensions d and consider training sets of different sizes. The approximation techniques are chosen by default by the rules described in Section 5.

- **GP:** Sample size $= 40, 80$; $d = 2$.
- **HDAGP:** Sample size $= 160, 320$; $d = 3$.
- **HDA:** Sample size $= 640, 1280, 2560$; $d = 3$.
- **SGP:** Sample size $= 2560$; $d = 3$.

Table 8.1 compares training times and errors of approximations obtained on the same computer with different settings of `Accelerator`. Times and errors are geometrically averaged over all the test problems. For the default value `Accelerator`=1, the actual averaged training times and errors are given. For the other values, the ratios $\frac{\text{reference value}}{\text{current value}}$ are given. We observe the clear general trend of increase of the error and decrease of the training time, though precise quantitative characteristics of the trend may depend significantly on test functions and approximation techniques.

	Technique Sample size	GP 40	GP 80	HDAGP 160	HDAGP 320	HDA 640	HDA 1280	HDA 2560	SGP 2560
`Accelerator` = 1	reference time, s reference error	0.45 9.7e-5	0.91 2.3e-5	22 5.6e-5	130 5.5e-5	740 9.5e-6	690 2.2e-6	1700 3.5e-8	850 9.8e-7
`Accelerator` = 2	time ratio error ratio	1.1 0.89	1.4 1.0	1.6 0.96	1.5 0.77	1.4 0.98	1.3 0.99	1.1 0.97	1.7 0.96
`Accelerator` = 3	time ratio error ratio	1.2 0.81	1.8 0.56	1.8 0.74	1.8 0.79	1.5 0.23	2.1 0.53	4.1 0.43	2.6 0.45
`Accelerator` = 4	time ratio error ratio	1.2 0.015	2.2 0.040	3.8 0.26	3.0 0.61	2.4 0.14	4.1 0.23	4.7 0.42	7.3 0.17
`Accelerator` = 5	time ratio error ratio	1.3 0.010	2.4 0.0068	4.0 0.26	3.0 0.61	2.5 0.14	19 0.025	9.8 0.39	21 0.021

Table 8.1: Effect of `Accelerator` on training time and accuracy. For the default value `Accelerator`=1, the actual averaged training times and errors are given. For the other values, the ratios $\frac{\text{reference value}}{\text{current value}}$ are given, where reference value corresponds to `Accelerator`=1.

Chapter 9

Accuracy Evaluation

9.1 Overview

Accuracy Evaluation (AE) is a part of **GTApprox** whose purpose is to estimate the accuracy of the constructed approximation at different points of the design space. If AE is turned on, then the constructed model contains, in addition to the approximation $\widehat{f} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, the AE prediction $\sigma : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}_{+}^{d_{\text{out}}}$ and the gradient of AE $\nabla\sigma : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ (see Section 2 and Figure 2.2).

AE is turned on or off by the option `AccuracyEvaluation` (default value is `off`).

AE prediction is performed separately for each of the d_{out} scalar outputs of the response. In the following, it is assumed for simplicity of the exposition and without loss of generality that the response has a single scalar component ($d_{\text{out}} = 1$).

The value $\sigma(X)$ characterizes the expected value of the approximation error $|\widehat{f}(X) - f(X)|$ at the point X. A typical example of AE prediction is shown in Figure 9.1. An approximation b) is constructed using a training set with DoE shown in a). The AE prediction is shown in d); also, its level sets are shown in c). We see that, on the whole, AE predicts larger error at the points further away from the training DoE. Also, the predicted error is usually larger at the boundaries of the design space. See Section 9.2 for more examples, including a workflow with repeated applications of AE.

The following notes explain the functionality of AE in more detail.

- The AE prediction is only an "educated guess" of approximation's error. As explained in Section 7.3, it is not possible, in general, to predict the exact values of the error. AE predictions should by no means be considered as a guarantee that the actual error is restricted to certain limits.

- The AE prediction is usually more efficient in terms of the *correlation* with the actual approximation errors rather than *matching* these errors. In other words, AE predictions are more relevant for estimating *relative* magnitude of error at different points of the design space. See examples further in this chapter.

- In general, the AE prediction σ reflects both of the two sources of the error:
  - uncertainty of prediction;
  - discrepancy between the approximation $\widehat{f}$ and the training data, resulting from smoothing and a lack of interpolation.

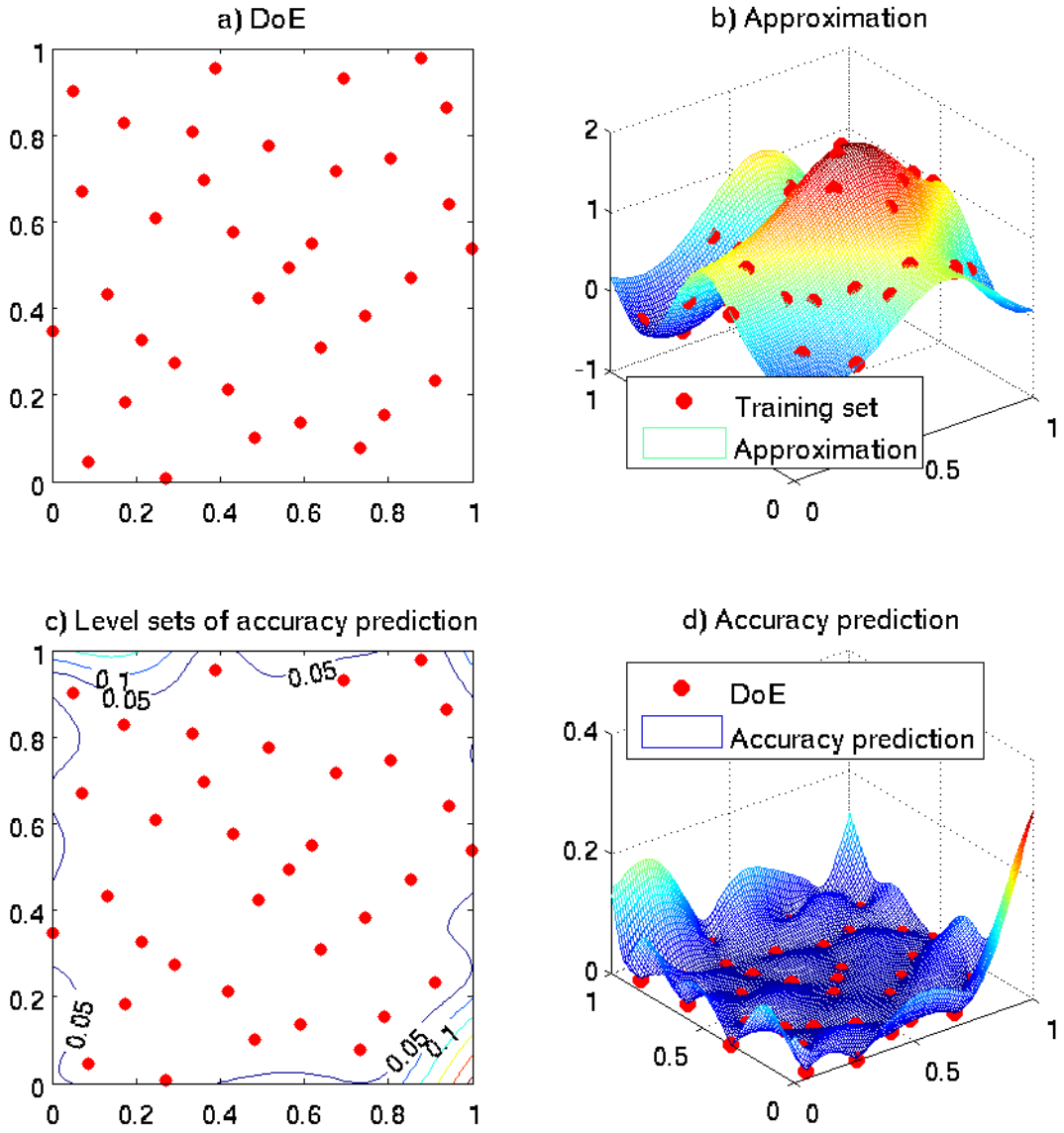


Figure 9.1: An example of accuracy prediction in **GTApprox**. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

The latter factor is present only for non–interpolating approximations.

- The prediction σ has the following general properties:
  - σ is a continuous function;
  - if the approximation $\widehat{f}$ is interpolating, then at the training DoE points X_k holds $\sigma(X_k) = 0$;
- AE is available in **GTApprox** for the following approximation techniques: SPLT, GP, HDAGP, SGP (see Section 4).
- The specific algorithms of error prediction depend on particular approximation techniques:
  - Gaussian-Process–based techniques (GP, HDAGP, SGP) estimate the error using the standard deviations of the Guassian process at particular points, see [26].
  - For Splines with Tension, the error is estimated by combining two different approaches:
    - comparison of the constructed spline with splines of a lower order;
    - using as a measure of error the distance to the training DoE, rescaled by an overall factor determined via cross–validation.
- The default technique for very large training sets (> 10000) in dimensions $d_{\text{in}} > 1$ is HDA (see Section 5), which means that, by default, AE is not available for such training sets. The user can choose GP or HDAGP as the approximation technique in such cases, but these techniques have high computer memory requirements, which may render the processing of the very large training set unfeasible. In fact, for large training sets the discrepancy $|\widehat{f} - f|$ tends to be a highly oscillating function (see, e.g., the example in Section 9.2.2), and the resolution and reproduction of these oscillations are computationally expensive. Another option, if AE is required, is to use SGP, which is available by default for sample sizes between 1000 and 10000. This technique trains the approximation using only a properly chosen subset of a manageable size in the whole training set.

9.2 Usage examples

9.2.1 AE with interpolation turned on or off

In this section we compare performance of AE in a noisy 1D problem with interpolation turned on or off. We select GP as the approximation technique as explained in Section 5.1. GP supports the interpolation mode but is not necessarily interpolating by default, see Section 6.2. (Note that the default 1D technique is SPLT which is always interpolating in **GTApprox**, see Section 4.1.2.)

The training data is noisy, and the DoE is gapped. The result of applying **GTApprox** with interpolation mode turned on or off is shown in Figure 9.2. Note that the forms of the approximation as well as the error prediction are quite different in these two cases. If the interpolation mode is turned on, then the tool perceives the local oscillations as an essential feature of the data and accurately reproduces the provided values. Accordingly,

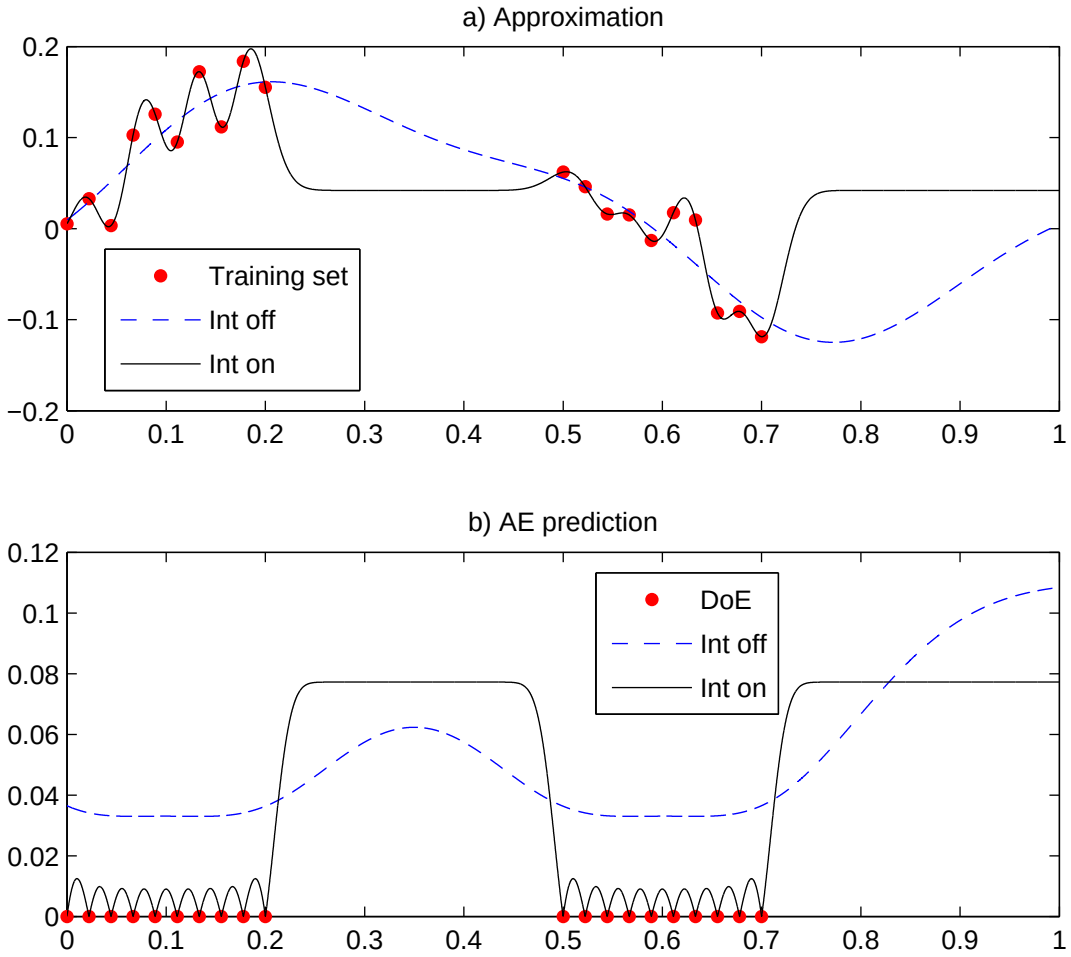


Figure 9.2: AE predictions with interpolation on or off. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

the predicted error vanishes at the points of the training DoE. In the domain where these points have a higher density, the predicted error is low, but in the DoE's gaps it is large.

In contrast, in the default, non–interpolating mode, the tool perceives the local oscillations as a noise and smooths the training data. The perceived uniformly high level of noise makes the tool add a constant to the error prediction, so that it is uniformly bounded away from zero. Another factor which affects the error prediction, the uncertainty growing with the distance to the training DoE, is also present here.

9.2.2 An adaptive DoE selection

Adaptive selection of DoE is a popular topic of current research, see, e.g., [12]. In this section we describe, for demonstration purposes, an example of the simplest adaptive process involving AE.

Consider the test function $f : [0,1]^2 \to \mathbb{R}$,

$$f(x_1, x_2) = 2 + 0.25(x_2 - 5x_1^2)^2 + (1 - 5x_1)^2 + 2(2 - 5x_2)^2 + 7\sin(2.5x_1)\sin(17.5x_1x_2).$$

The graph of this function is shown in Figure 9.3a). Figure 9.3b) shows the level sets of this function.

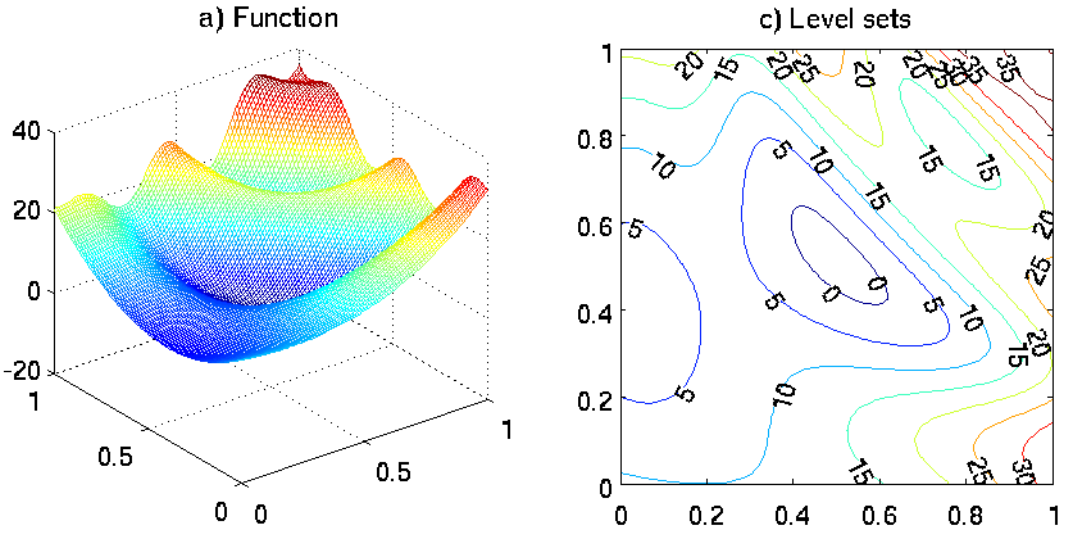


Figure 9.3: Adaptive DoE: the approximated function f

We start the adaptive DoE process by selecting an initial DoE. We choose as the initial DoE an Optimal Latin Hypercube Sample in $[0,1]^2$ generated by **GT DoE**. This set is shown in Figure 9.4a). Figures b)–f) show the approximation constructed using this DoE, the AE prediction and the actual error $|\widehat{f} - f|$. As the training set is very small, the approximation is not very close to the original function.

To achieve a better accuracy, we perform an adaptive expansion of the training set. This is an iterative procedure; a single iteration consists of the following steps:

1. Using the current DoE, an approximation with an AE prediction is constructed.
2. The obtained AE prediction is approximately maximized on the design space $[0,1]^2$:
   (a) a random set P of 10^4 points is generated in $[0,1]^2$;
   (b) the AE predictions are computed on the points P;
   (c) the point $p \in P$ with the largest AE prediction is selected.
3. The selected point p is added to the DoE.

By adding to the DoE the point with the maximum AE prediction, we attempt to reduce the uncertainty in the data in an optimal way. Note also that computing AE predictions on 10^4 points is quite fast, since AE is a part of the analytic model constructed by **GTApprox**. If adaptive sampling is applied to a very slow response function instead of the analytic function f, computation of AE prediction on 10^4 points will remain easily affordable.

We perform 70 iterations as described above, and reach the final state shown in Figure 9.5. The final DoE is shown in Figure 9.5a). The black dots are the initial 10 points, and the red ones are those added during the iterations. Note that the points tend to uniformly fill the design space, though their density is slightly higher near the borders, where uncertainty tends to be higher. The final approximation and its level sets are shown in b)–c), and they are now quite close to the original, shown in Figure 9.3.

Note that, as seen in Figures 9.5d),f), though the function f is reasonably simple, both predicted and actual errors are highly oscillating functions at large sizes of the training set.

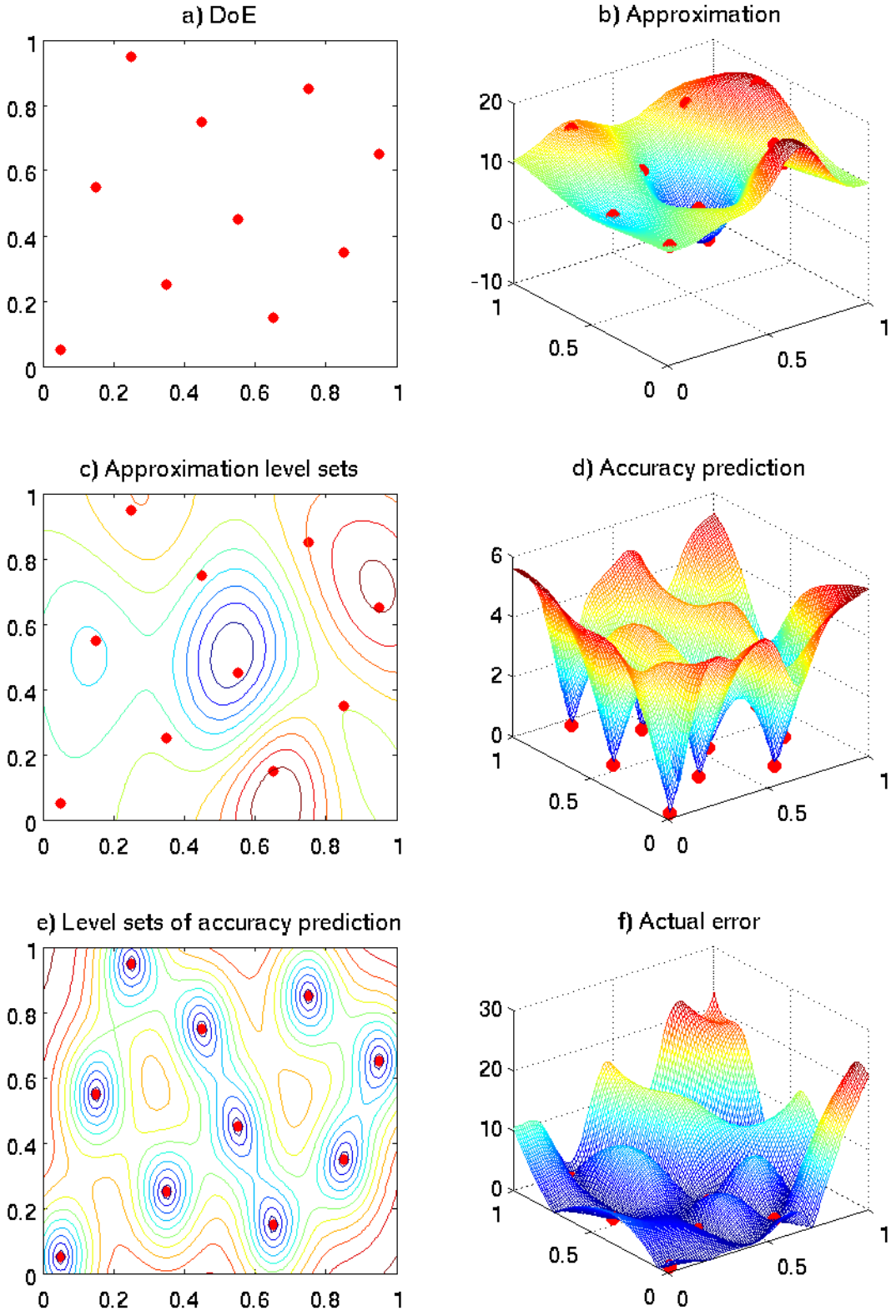


Figure 9.4: Adaptive DoE, initial state: a) DoE; b) approximation; c) approximation's level sets; d) AE prediction; e) level sets of the AE prediction; f) actual eapproximation error $|\widehat{f} - f|$.
Note: This image was obtained using an older MACROS version. Actual results in the current version may differ.

This is why it becomes computationally increasingly difficult to resolve the local detail of the error as the training DoE grows, and limits AE to moderately sized training sets.

Finally, in Figure 9.6 the evolution of the RMS error of the approximation is shown, as size of the training set increases from 10 to 80. The general trend of error decrease is clearly seen.

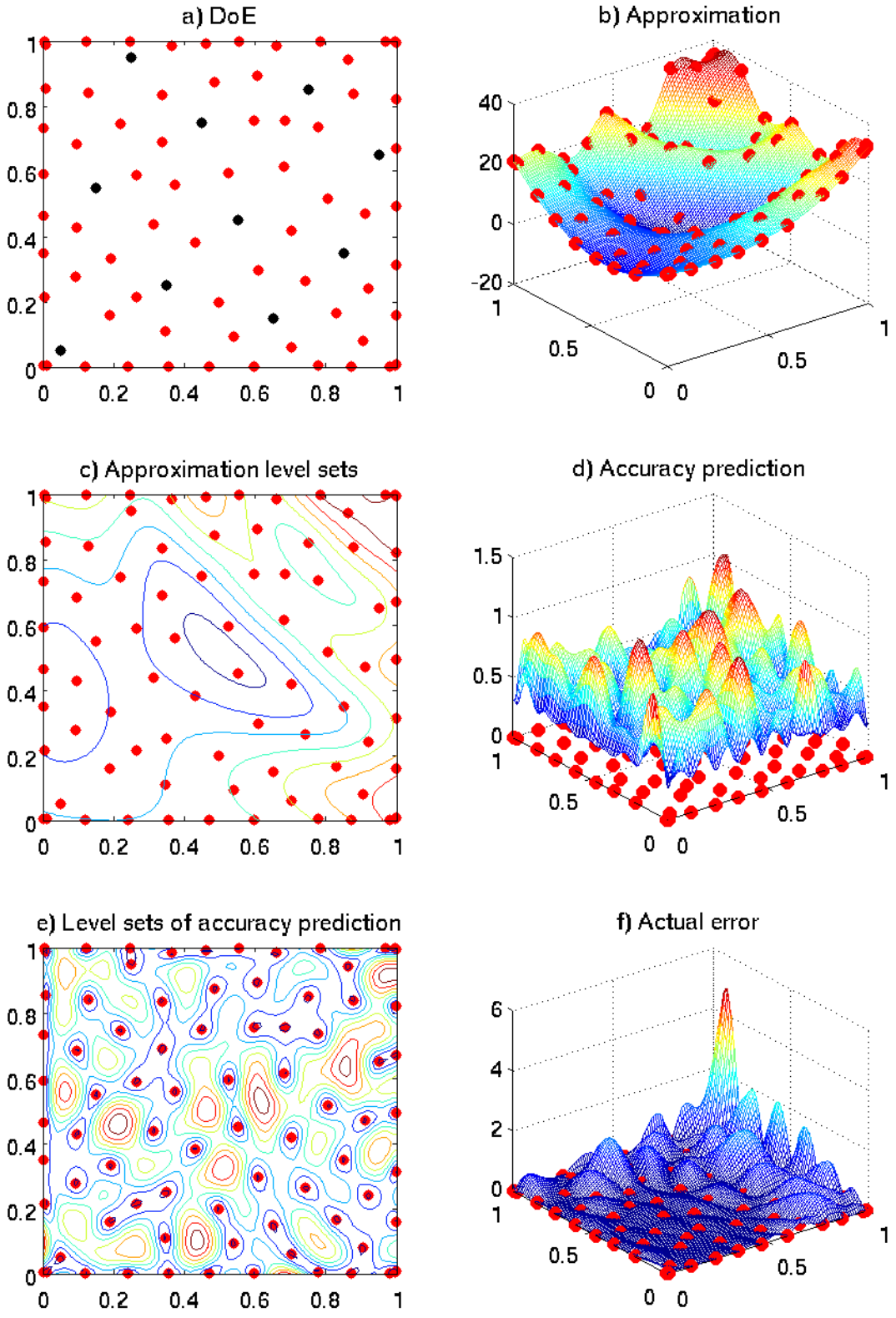


Figure 9.5: Adaptive DoE, final state. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

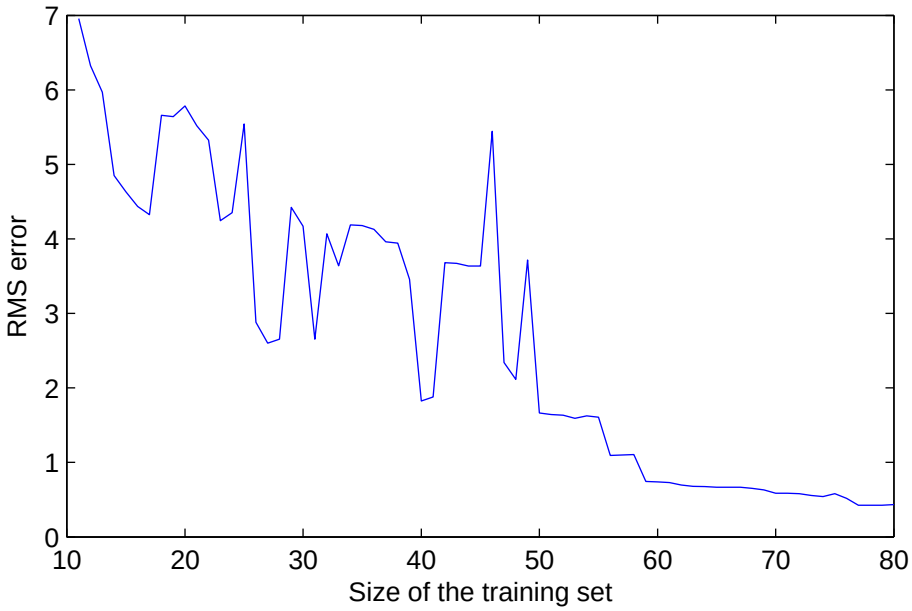


Figure 9.6: Adaptive DoE, error vs size of the training set. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

Chapter 10

Internal Validation

10.1 Description of the procedure

The Internal Validation (IV) procedure is an optional part of **GTApprox** providing an estimate of the expected overall accuracy of the approximation algorithm on the user–supplied training data. This estimate is obtained by a (generalized) cross–validation of the algorithm on the training data. Various properties of this procedure are discussed in Section 10.2.

The result of this procedure is a table with errors of several types, written to the log file and to the constructed model's info file.

The procedure can be turned on/off by the option `InternalValidation` (the default value is `off`).

The procedure depends on the following parameters which can be set up through the advanced options of **GTApprox**:

- The number N_{ss} of subsets that the original training set S is divided into, where $2 \leqslant N_{\text{ss}} \leqslant |S|$. This number is set by the option `IVSubsetCount`.
- The number N_{tr} of training/validation sessions, where $1 \leqslant N_{\text{tr}} \leqslant N_{\text{ss}}$. This number is set by the option `IVTrainingCount`.
- The seed for the pseudo–random division of S into subsets. This seed is set by the option `IVRandomSeed`.

The default values of the parameters N_{ss}, N_{tr} are given by

$$\begin{aligned} N_{\text{ss}} &= \min(10, |S|), \\ N_{\text{tr}} &= \left\lceil \min\left(|S|, \frac{100}{|S|}\right) \right\rceil, \end{aligned} \tag{10.1}$$

where $\lceil a \rceil$ denotes the smallest integer not less than a.

The Internal Validation accompanies the construction of the approximation. If IV is turned on, the model construction procedure includes the following steps:

1. From the options' values and the properties of the training set S, the tool determines the appropriate approximation algorithm $\mathcal{A}$ to be used when constructing the main approximation $\widehat{f}$ provided to the user.
2. After that, the tool starts the Internal Validation of the algorithm $\mathcal{A}$ on the sample S.

(a) The set S is randomly divided into N_{ss} disjoint subsets $(S_k)_{k=1}^{N_{\text{ss}}}$ of approximately equal size.

(b) For each $k = 1, \ldots, N_{\text{tr}}$, an $\mathcal{A}$-based approximation $\widehat{f}_k$ is trained on the subset $S \backslash S_k$, and its errors $E_{k,l}$ of one of the three standard types, MAE, RMS and RRMS (see below) are computed on the complementary test subset S_k, separately for each scalar component $l = 1, \ldots, d_{\text{out}}$ of the output.

(c) The cross–validation errors $(E_l^{\text{cv}})_{l=1}^{d_{\text{out}}}$ are computed as the *median* values of the errors $E_{k,l}$ over the training/validation iterations $k = 1, \ldots, N_{\text{tr}}$.

(d) The total cross–validation errors $E_{\text{mean}}^{\text{cv}}, E_{\text{rms}}^{\text{cv}}, E_{\text{max}}^{\text{cv}}$ are computed as the *mean*, *root-mean-squared* or *maximum* values, respectively, of the cross–validation errors $(E_l^{\text{cv}})_{l=1}^{d_{\text{out}}}$ over the output components $l = 1, \ldots, d_{\text{out}}$.

3. The obtained errors $(E_l^{\text{cv}})_{l=1}^{d_{\text{out}}}$ and $E_{\text{mean}}^{\text{cv}}, E_{\text{rms}}^{\text{cv}}, E_{\text{max}}^{\text{cv}}$, where E is MAE, RMS or RRMS, are written to the log file.

4. Finally, the main approximation $\widehat{f}$ is trained using all the training data S and saved in the constructed model.

In **GTApprox**, the following types of errors are computed during Internal Validation:

- Mean absolute error,

$$\text{MAE}_{k,l} = \frac{1}{|S_k|} \sum_{(X,Y) \in S_k} \left| \widehat{f}_k^{(l)}(X) - Y^{(l)} \right|,$$

  where $Y^{(l)}$ is the l'th output component of the training point (X, Y).

- Root-mean-squared error,

$$\text{RMS}_{k,l} = \left(\frac{1}{|S_k|} \sum_{(X,Y) \in S_k} \left| \widehat{f}_k^{(l)}(X) - Y^{(l)} \right|^2 \right)^{1/2}$$

- Relative root-mean-squared error,

$$\text{RRMS}_{k,l} = \frac{\left(\frac{1}{|S_k|} \sum_{(X,Y) \in S_k} \left| \widehat{f}_k^{(l)}(X) - Y^{(l)} \right|^2 \right)^{1/2}}{\left(\frac{1}{|S_k|} \sum_{(X,Y) \in S_k} \left| \overline{Y^{(l)}}^{S_k} - Y^{(l)} \right|^2 \right)^{1/2}}, \tag{10.2}$$

  where $\overline{Y^{(l)}}^{S_k}$ is the mean of $Y^{(l)}$ on the test subset S_k.

10.2 Notes and Remarks

- Cross-validation is a well-known and well-established way of statistical assessment of the algorithm's efficiency on the given training set (see, e.g. [1, 21]). It should be stressed, however, that it does not directly estimate the predictive power of the main approximation $\widehat{f}$ *outside* the training set S. Rather, the purpose of cross-validation is to assess the efficiency of the approximation algorithm $\mathcal{A}$ on various subsets of the available data, assuming that the conclusions can be extended to the (unavailable) observations from the total design space.

- The parameter N_{ss} controls the number and the size of test subsets. The parameter N_{tr} is the total number of internal training/validation sessions. The parametrization of the IV procedure by N_{ss} and N_{tr} endows the procedure with sufficient flexibility: by varying these parameters one can obtain, as special cases, a number of standard validation types:
  - $N_{\text{ss}} = |S|$ corresponds to the leave-one-out cross-validation, with N_{tr} points to leave out. Setting $N_{\text{tr}} = N_{\text{ss}} = |S|$ yields the standard full leave-one-out cross-validation.
  - $N_{\text{tr}} = 1$ corresponds to a single instance of training and validation, where the initial set S is divided into a training and a test subsets $S_{\text{train}}, S_{\text{test}}$ so that $\frac{|S_{\text{train}}|}{|S_{\text{test}}|} \approx \frac{N_{\text{ss}}-1}{1}$.

  In general, only part of the subsets S_k are used as test subsets during cross–validation.

- If the size of the training set is not an exact multiple of N_{ss}, then the sizes of the subsets S_k are chosen as nearest integers to $|S|/N_{\text{ss}}$ from above or below, e.g., if $|S| = 23$ and $N_{\text{ss}} = 10$, then the sizes would be 3, 3, 3, 2, 2, 2, 2, 2, 2, 2.

- The relative root-mean-squared error (10.2) shows the accuracy of the approximation algorithm $\mathcal{A}$ relative to the accuracy of the trivial approximation by a constant, computed as the mean value of the response on the training subset. The value of RRMS close to 1 shows that on the given training set the approximation algorithm $\mathcal{A}$ is approximately as efficient as just predicting the mean value. This is usually due to the lack of training data or a too noisy or complex response function.

- One should remember that, as all relative errors, the RRMS error is unstable due to potential vanishing of the denominator in formula (10.2). This can sometimes occur because of coincidental matches between $\overline{Y^{(l)}}^{S_k}$ and $Y^{(l)}$. It is recommended to keep the sets S_k sufficiently large when considering IV with the RRMS error, because otherwise the chances of the denominator to vanish for some k, l significantly increase. Computing the median value, rather than the mean value, at step 2c of the IV procedure is intended to remove outliers and decrease the effect of such coincidental matches.

- The *maximum* error is not included in the list of errors computed during cross-validation (MAE, RMS, RRMS) because its estimates are usually less robust. Note, e.g., that if the response function f happens to be unbounded on the design space, i.e., one can find design points with arbitrarily large absolute values of the response function, then the maximum approximation error is definitely infinite, while the IV–estimated errors, though possibly large, are definitely finite, since they are computed using finitely many finite training data. Such disagreement is also possible for mean and RMS errors, but it is less likely, since it requires a sufficiently strong singularity of the response function (note that the errors are always ordered MAE $\leqslant$ RMS $\leqslant$ MAX so that RMS may be finite if MAX is infinite, but not the other way round).

  Note also that outliers outside the training data cannot be predicted based on this data. Yet, an outlier can make a significant impact on the maximum error. At the same time, the effect of a single outlier on the mean or RMS error on a test set is usually much smaller or even negligible, as it is divided by the size of this test set.

- Since the training conditions for the approximations $\widehat{f}_k$ are comparable to those for the main approximation $\widehat{f}$, turning Internal Validation on will approximately increase the total time required by **GTApprox** to construct a model by a factor of $N_{\text{tr}} + 1$.

- The reliability of cross–validation is normally expected to increase with the amount N_{tr} of verifications performed. The direct effect of N_{ss} on the reliability of cross–validation is less certain: on the one hand, increasing N_{ss} makes subsets S_k smaller and hence brings the training subsets $S\backslash S_k$ closer to the original training sample S; on the other hand, the validation results of the approximations $\widehat{f}_k$ on the test subsets S_k are more reliable if these test subsets are larger. However, choosing a larger N_{ss} also increases the largest possible value of N_{ss}. The most reliable type of cross–validation is expected to be the full leave-one-out cross–validation (except for the RRMS error, see above), but it is also the longest.

- The default values (10.1) for N_{ss} and N_{tr} are selected so as to balance the runtime of the tool with the reliability of cross–validation. If the training sample is small, then the training time for a single approximation is relatively short and one can usually afford a larger N_{ss}. On the other hand, if the training sample is large, then one usually expects more reliable results from a single validation, so that one can set $N_{\text{tr}} = 1$. The default values (10.1) are chosen so that:

  - the full leave-one-out cross–validation is performed at $|S| \leqslant 10$;
  - a single training/validation is performed at $|S| \geqslant 100$;
  - the formula continuously interpolates between the above two extreme modes at $10 < |S| < 100$;
  - the size of the training subset $S\backslash S_k$ at each iteration is approximately $0.9|S|$;
  - the total number of points in the test subsets over which the validation is performed is approximately 10 at $|S| < 100$ or $|S|/10$ at $S \geqslant 100$.

  If the accuracy assessment has a higher priority than the runtime in a given application, then the user should increase the values of N_{ss} and N_{tr}.

  Note, however, that, regardless of the settings of the IV procedure, its efficiency for any conclusions related to points outside the training sample, or to the total design space, is fundamentally restricted by the total amount of information contained in the training set and the actual validity of the inference assumptions.

- Since only subset of training sample is used for approximation construction during Internal Validation , turning `InternalValidation = On` will change the restriction on minimal sample size $|S|$ (see details in the technical reference [15]).

- Now **GTApprox** provides a way to save outputs calculated during IV procedure. To save predictions calculated with constructed during IV surrogate models one has to turn `GTApprox/IVSavePredictions` options `on`. Obtained predictions can be used, for example, to get scatter plot of true values and predictions (see subsection 10.3.3). Default value for `GTApprox/IVSavePredictions` option is `off`.

It should be stressed that though cross-validation is a conventional way to assess the efficiency of the approximating method on the given data, a precise estimate of the errors on

the whole design space can only be obtained by an external validation of the approximation $\widehat{f}$ itself, i.e., by testing it on a sufficiently large and properly selected test sample. The task of selecting such a test sample is beyond the functionality of **GTApprox**. However, if the user already has a test sample, applying the approximation to it and computing the desired accuracy characteristics are straightforward operations, and the user should have no difficulty in performing them.

10.3 Usage examples

10.3.1 Internal Validation for a simple and a complex functions

In this section we apply Internal Validation to two well–known two-dimensional test functions. The first is the Rosenbrock function,

$$f_1(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

and the second is the Rastrigin function,

$$f_1(x_1, x_2) = 20 + (x_1 - 10\cos(2\pi x_1)) + (x_2 - 10\cos(2\pi x_2)).$$

We consider the Rosenbrock function on the design space $[-2.048, 2.048]^2$ and the Rastrigin function on the design space $[-5.12, 5.12]^2$. It will be convenient, however, to map the two design spaces, by rescaling, to the same unit cube $[0, 1]^2$, i.e., we set $x_k = -2.048 + 4.096x'_k, k = 1, 2$, in the first case and $x_k = -5.12 + 10.24x'_k, k = 1, 2$, in the second, and consider the two functions as functions of the variables $0 \leqslant x'_1, x'_2 \leqslant 1$.

We will perform Internal Validation of the approximation algorithm on these test functions, with a common training set chosen as a Latin Hypercube Sample of size 40, provided by **GT DoE**.

The graphs of the two functions are shown in Figure 10.1, a) and b), and the training set is shown in Figure 10.1, c). Note that the Rosenbrock function is a quite simple function, and we expect that 40 training points should be sufficient for a rather accurate approximation. On the other hand, the Rastrigin function is a complex multimodal function with about 100 local minima/maxima. It is clear that 40 training points are far too few to resolve the high frequency oscillations and hence achieve any reasonable approximation accuracy in this case.

We perform Internal Validation with default settings. According to formulae 10.1, the training subsets then have size 36, the test subsets have size 4, and there are 3 training/validation iterations, for each of the two models.

The results of the IV procedure are shown in Table 10.1. These results correctly reflect our expectations. In the case of the Rosenbrock function, the RRMS error is significantly less than 1, which confirms that the available training data allow to approximate the function very accurately. On the other hand, the RRMS error for the Rastrigin function is approximately 1, which shows that the training data is insufficient, and the advanced approximation algorithm offers no improvement over the trivial approximation by the mean value.

10.3.2 Comparison of Internal Validation and validation on a separate test set

In this section we consider again the Rosenbrock and Rastrigin functions, and compare the results of Internal Validation with the approximation errors obtained using an independent test sample. To speed up the tests, GP is used as the approximation technique (see Section 4).

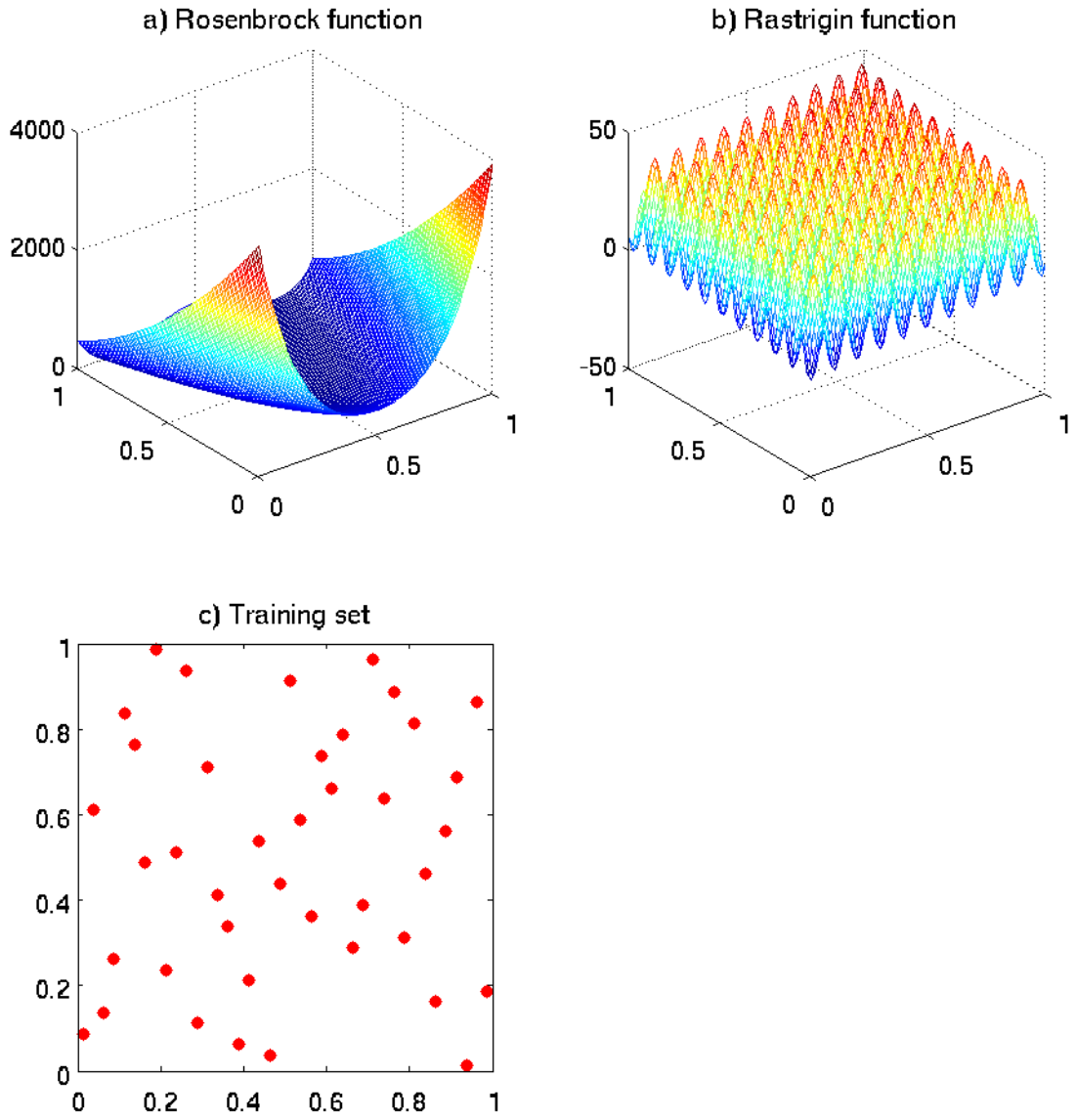


Figure 10.1: a) Rosenbrock function; b) Rastrigin function; c) Training set

	Rosenbrock	Rastrigin
MAE	0.027188	5.7805
RMS	0.032297	7.3507
RRMS	7.2848e-05	1.2256

Table 10.1: Internal Validation results for the Rosenbrock and Rastrigin functions

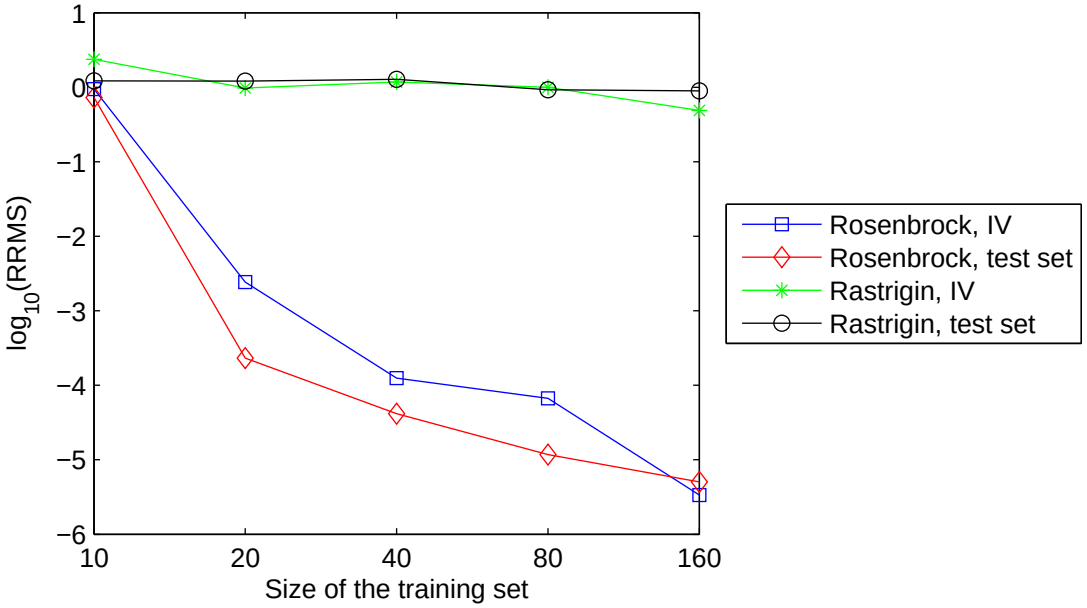


Figure 10.2: Comparison of Internal Validation and validation on a separate test set. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

We consider an increasing sequence of Sobol training sets in the design space $[0,1]^2$ obtained using **GT DoE**. The sizes of the training sets are

$$10, 20, 40, 80, 160.$$

In each case, we apply Internal Validation with default settings to the Rosenbrock and Rastrigin functions. The corresponding RRMS errors on the logarithmic scale are shown in Figure 10.2.

In parallel to Internal Validation, we consider the main approximations $\widehat{f}$ constructed by **GTApprox** with the above training sets, and compute their RRMS errors on the test set which is selected as the uniform 100×100 grid in $[0,1]^2$. The resulting errors are also shown in Figure 10.2.

For both test functions, we observe a general agreement between the Internal Validation and the validation on a separate test set. In the case of the Rastrigin function, both methods yield an RRMS error which remains close to 1 up to the largest size of the training set. As pointed out in the previous section, this is due to the high complexity of the function.

In contrast, approximation accuracy for the Rosenbrock function rapidly grows with the size of the training set, which is reflected by both methods of accuracy estimation. The exact values of the errors obtained by the two methods are not equal, but the overall trends are quite close.

Note that the Rosenbrock function is very similar to the oscillating function of Section 7.3 illustrating the limitations of the error estimating procedures. As explained there, a DoE in the form of a regular grid leads to misleading results for functions with high–frequency periodic features like the Rastrigin function, so if we used such a DoE instead of the Sobol sequence in this example, then IV would have underestimated the actual error. Another factor which has made IV results more reliable in this example is the repeated application of IV, at several instances of the growing DoE.

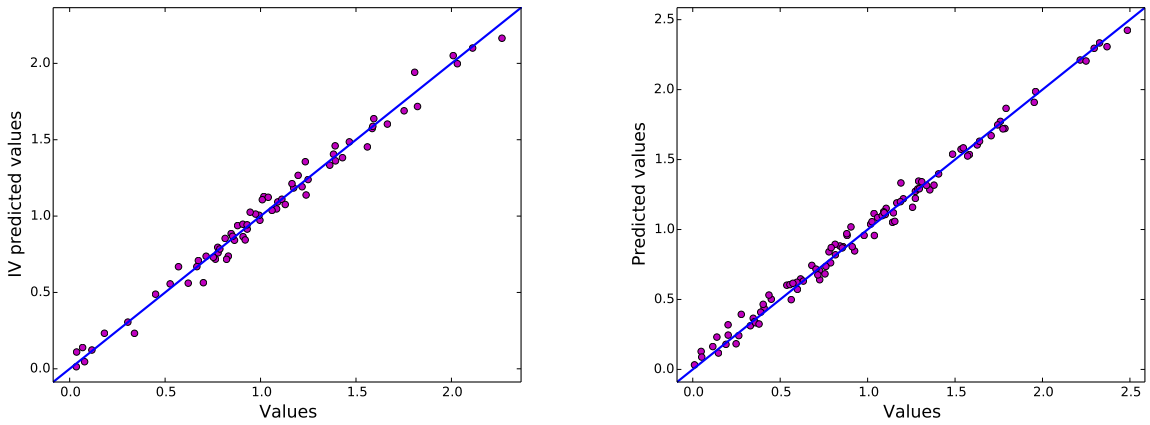
10.3.3 Scatter plot obtained with IV predictions

We consider a problem of noisy function approximation:

$$f(X) = \sum_{i=1}^{d_{\text{in}}} x_i^2 + \varepsilon,$$

where ε is a Gaussian white noise with zero mean and variance 0.05^2. Input dimension $d_{\text{in}} = 3$. Training sample of size 70 consists of points uniformly distributed in hypercube $[0, 1]^3$.

A scatter plot obtained with IV predictions is provided in figure 10.3a. In this case IV predictions are close to the true values, so obtained model seems to be accurate enough. RRMS value estimated with Internal validation equals to 0.116. For the sake of comparison we obtain a scatter plot and RRMS error for a separate test sample. A scatter plot is in figure 10.3b and RRMS value for a separate test sample equals to 0.091. One can see that plots and errors obtained with Internal validation and with a separate test sample are close to each other. See script `plotScatterIVpredictions.py` for additional details.



(a) Scatter plot obtained with Internal validation out-put

(b) Scatter plot obtained with a separate test sample

Figure 10.3: Scatter plot for IV outputs

Chapter 11

Processing user-provided information

It's a common situation when there is some problem-specific information that can be used during surrogate model construction. The processing procedure is a way to transform engineering knowledge about the specific problem into recommendations of the **GTApprox** usage and additional insights into the data. Two main goals of the procedure are to give an advice to user on how to build an accurate surrogate model using given data and how to improve an existing surrogate model. Also, the processing provides additional information about data, which can be useful in some situations.

One can run processing procedure in two modes: preprocessing and postprocessing. The best time to apply the preprocessing mode is strictly before the first run of **GTApprox** for the considered problem. During *preprocessing step* the preprocessing procedure performs some analysis of the given data and treats user-defined options for preprocessing (these options defines peculiarities of the problem, which can be pointed by the user). Obtained results are used to compose a list of recommendations on **GTApprox** running options. The second mode examines given surrogate model using train set (and test set, if available). During *postprocessing step* the postprocessing procedure analyses given data and a constructed surrogate model, treats user requirements and wishes concerning desired properties of surrogate model. Output in this case is a set of recommendations on how to improve an already constructed surrogate model and additional data analysis.

The later sections describe in more details workflow for the preprocessing and postprocessing modes.

11.1 Preprocessing

In this case input contains a data set of points, corresponding function values and user's response in the form given below. The workflow of the preprocessing mode consists of a couple of steps:

1. Analyse given data.
2. Treat user's response for the preprocessing mode.
3. Create recommendations.

The first two steps are described in details below. The last step of preprocessing workflow summarizes results of its work in two ways. First, it creates recommended set of options which are most suitable for the given data and user requirements. Also, a report concerning

the given data is created with expanded version of answers to allow a user to select options reasonably. Examples of recommended options for some of the given questions are described below. These examples describe how the tool is supposed to work in some cases.

11.1.1 Analysis step

During the analysis step, **GTApprox** processes raw input data, preparing it to be used as a training sample and also exploring its properties.

Let **XY** denote the matrix containing the training set (see Chapter 2 for its detailed description). With this matrix as an input, **GTApprox** performs the following procedures:

1. **Data cleaning. GTApprox** removes all rows containing `NaN` or `Inf` values from **XY**. Non-numeric values can not be used when training a model, thus such points have to be removed from the training set.

2. **Redundant data removal. GTApprox** checks if **XY** contains duplicated rows and removes the duplicates. A duplicate row means that the same point is found in the training set twice (or more). Every instance of such point except the first one does not provide any useful information and thus should be ignored.

As a result of the two procedures above, we obtain a reduced matrix $\widetilde{\mathbf{XY}}$ containing only correct data. Let $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{Y}}$ further denote its submatrices containing the data for input and output components (similarly to **XY**).

3. **Data structure analysis.** Includes the following tests:

   - Test for Cartesian structure. **GTApprox** checks if $\widetilde{\mathbf{X}}$ has (full or incomplete) Cartesian product structure (see Chapter 12 for details). This test may be turned off by setting the `TensorStructure` hint to `no`.
   - Test for uniform distribution. **GTApprox** checks if the points in $\widetilde{\mathbf{X}}$ are uniformly distributed in the design space. This test is based on the notion of discrepancy, a special measure of uniformity in multidimensional spaces.

4. **Dependency type detection. GTApprox** searches for linear and quadratic dependences in the input data. This is done by constructing simple linear and quadratic models and checking if the model values in the points from $\widetilde{\mathbf{X}}$ are close enough to values in $\widetilde{\mathbf{Y}}$.

11.1.2 Processing user responses

User-defined preprocessing options contain answers on a sequence of multiple-choice questions about the data. The list of questions (hints) and their short description are given in the Technical Reference. All the hints are formalized as options `GTApprox/Preprocess/HintName`.

Hints can be decomposed in three groups:

- General hints: these hints describe the user's opinion on sample data.
- Requirement hints: these hints set specific requirements for the model to be built.

- Preprocessing settings: these hints are used for preprocess function tuning.

As a result of hints' analysis special recommendations on run options of **GTApprox** are made.

11.1.3 Example

The target function to be approximated is a linear function in $\mathbb{R}^{10}$:

$$f(\vec{x}) = \sum_{i=1}^{10} x_i.$$

We observe noisy function values $f(\vec{x})+\varepsilon$, where ε is zero-mean Gaussian random variable with variance equal to 0.05. The linear nature of function is therefore hidden but can be detected with the help of preprocessing. Let's consider a learning sample of size 25. We generate a sample from the uniform distribution for the $[0,1]^{10}$ hypercube.

For a given sample size and dimension GP technique is automatically selected by **GTApprox**. But it can be not the best one for the given linear problem. The GP technique has highly nonlinear nature and it can fail to detect linear properties in case of small sample and noisy observations.

A good solution to this question is to run preprocessing before model construction to detect possible linearity. Options recommended by preprocess function contain suggestion to use other technique: `linear` mode of RSM. So, the linear nature of the problem was successfully detected.

If we compare errors of approximation on test set of 1000 points, we can see significant benefit of using options recommended by preprocess function.

```
Error of approximation obtained by initial model:   2.16.
Error of approximation obtained by preprocessed model:   0.88.
```

11.2 Postprocessing

Input for postprocessing function consists of a surrogate model, data (a training set and, if available, a test set) and user's response in the form given below (also include data analysis options). Postprocessing workflow steps are listed below:

1. Analyse the input.
2. Treat user's response for the postprocessing mode.
3. Create recommendations list.

11.2.1 Postprocessing analysis

During *Analysis step* the tool tries to estimate model quality. User specifies options of the data analysis. List of possible steps is given below:

- calculate the given surrogate model approximation errors;
- calculate constant approximation errors;

- calculate linear regression model approximation errors;
- calculate accuracy evaluation errors (if accuracy evaluation is available);
- run Internal Validation (see chapter 10) for the training set and calculate internal validation errors.

Errors are calculated for the training set and, if available, for the test set. Errors include MAE, Max, Min, RMS, RRMS (see chapter 7). In addition, for accuracy evaluation the function calculates Pearson correlation coefficient (if accuracy evaluation available). Internal Validation seems to be the most accurate way (in case there is no test set) to discover if selected method to build a surrogate model suits the given data, but it is also a time-consuming procedure. Note, that to run Internal Validation one has to specify corresponding option.

Calculated errors are saved in the model, so one can access them in a simple way. This information can be crucial for determining if surrogate model is good enough.

Postprocessing saves selected number of points with the biggest errors. Number of points is specified by the user directly or determined using the quantile value specified by the user (so we select points with the surrogate model errors below the given quantile for the full set of points). The user can specify both number of selected points and quantile value. Points are selected and saved separately for each output dimension and for training and test samples. Corresponding values and indexes are also presented in the output of the function. Points with the biggest errors can be very important for the user for analysis of approximation behaviour and quality.

The abovementioned features in the postprocessing function are tuned by specifying postprocessing hints. Full list of hint names and their values is found in the MACROS for Python Technical Reference.

11.2.2 User's opinion treatment

After the analysis step postprocessing treats hints which specify user's responses to the list of questions. Responses define wide range of possible problems concerning the constructed surrogate model. The list of hint names and their values is found in the MACROS for Python Technical Reference.

The general way to create a recommendation is to look directly what user wants and try to meet this requirement. User's responses are used to create a recommendations list. Recommendations are prioritized. Corresponding answers will be given on basis of methodological considerations.

The postprocessing function usage example is presented below.

11.2.3 Example

Smoothness example

The target function we approximate is a step function in $\mathbb{R}^2$:

$$f(X) = \begin{cases} 1, & \text{if } \sum_{i=1}^{d_{in}} x_i > 0, \\ 0, & \text{else.} \end{cases}$$

The function is depicted at the figure 11.1.

It is easy to note, that this function has a discontinuity along the line $\sum_{i=1}^{d_{in}} x_i = 0$. The user knows about presence of discontinuity in the function, but he doesn't know how to select appropriate options in this case.

In some cases it it desirable to keep this discontinuity in a surrogate model $\hat{f}(X)$, but in some cases discontinuity in a surrogate model can be a problem. Also, incorporating prior knowledge about the problem can increase resulting surrogate model quality.

Really, consider a sample with size $|S| = 100$ and $d_{in} = 2$. We generate a sample from the uniform distribution in $[-1, 1]^2$.

In case we build a surrogate model with default parameters `GP` technique is selected automatically by the decision tree, and a surrogate model approximation will be like the one depicted in the figure 11.2. One can see, that this surrogate model tries to smooth data i.e. there is no discontinuity any more. Also, surrogate model quality isn't very good.

So, one wants a quick solution which suites her/his requirement that the discontinuity should be kept by the surrogate model. A good solution to this question is to run postprocessing which is based on the already constructed surrogate model. The problem is that the approximation is oversmoothed, so we set `GTApprox/Postprocess/Smoothness` to 1. Recommended options consist an advice to use other technique, `HDAGP`. We run **GTApprox** using options, recommended by the postprocessing function.

Approximation for the surrogate model, constructed using recommended options, is depicted in the figure 11.3. One can note a couple of things about this approximation:

- The approximation preserves desired discontinuity.
- This model fits the training data much better than the default one.

The postprocess function has solved the presented problem.

Worst points example

Another useful functional for postprocessing is access to points with biggest approximation errors. We consider a data set based on the real data from [13] which consists of variables based on physicochemical tests (fixed acidity, density, sulphates, etc.) and wine quality. The problem is to build a surrogate model for wine quality using other presented variables.

Training set size $|S|$ equals 133, number of input variables d_{in} equals 11. Problem has rather high dimensionality comparing to the training sample size, so it is desirable to use simple method for surrogate model construction. We use regularized linear regression technique available in **GTApprox** (see chapter 4).

For the training sample we build a surrogate model and then run postprocessing. Scatter plot for true values and values obtained using the surrogate model is presented in Figure 11.4. One can see, that there are a couple of points with big approximation errors. By looking inside the data one can see that among this points their corresponding wines are mostly red wines. The test set contains only white wines, the training set contains only a couple of red wines and all these red wines belongs to the points with the biggest approximation errors.

We exclude these red wines from the training sample and construct a new surrogate model using censored training set. Comparison of errors for the test set is presented in Table 11.1. One can see that using censorship of the training set we significantly reduce surrogate model errors.

In the presented example we gain profitable insight into the data using postprocessing. By excluding of redundant points from the training sample we significantly increase approximation quality.

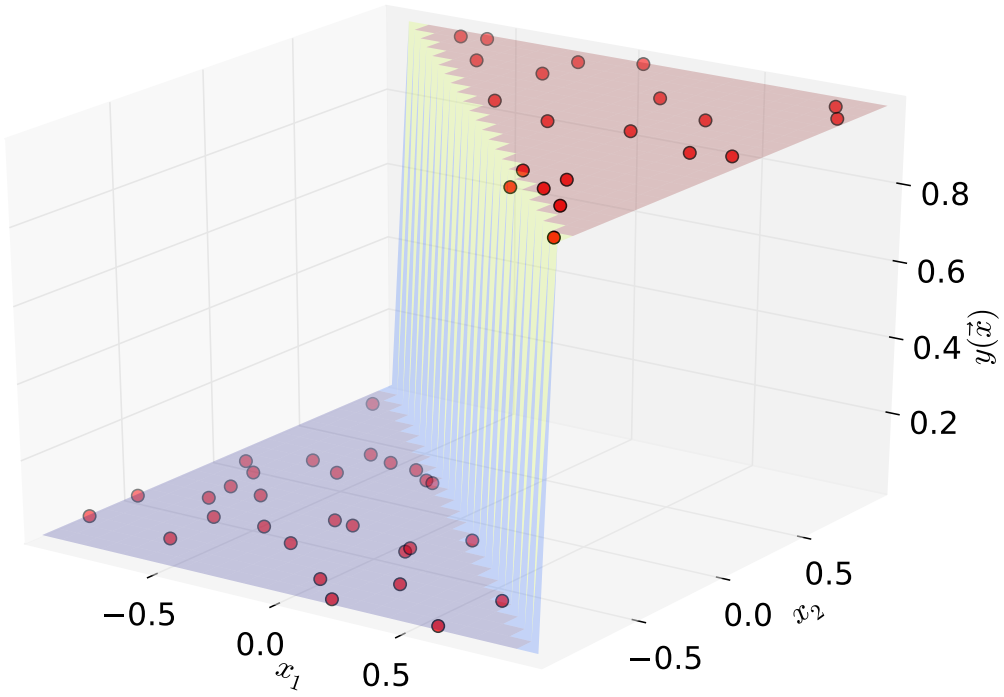


Figure 11.1: True function

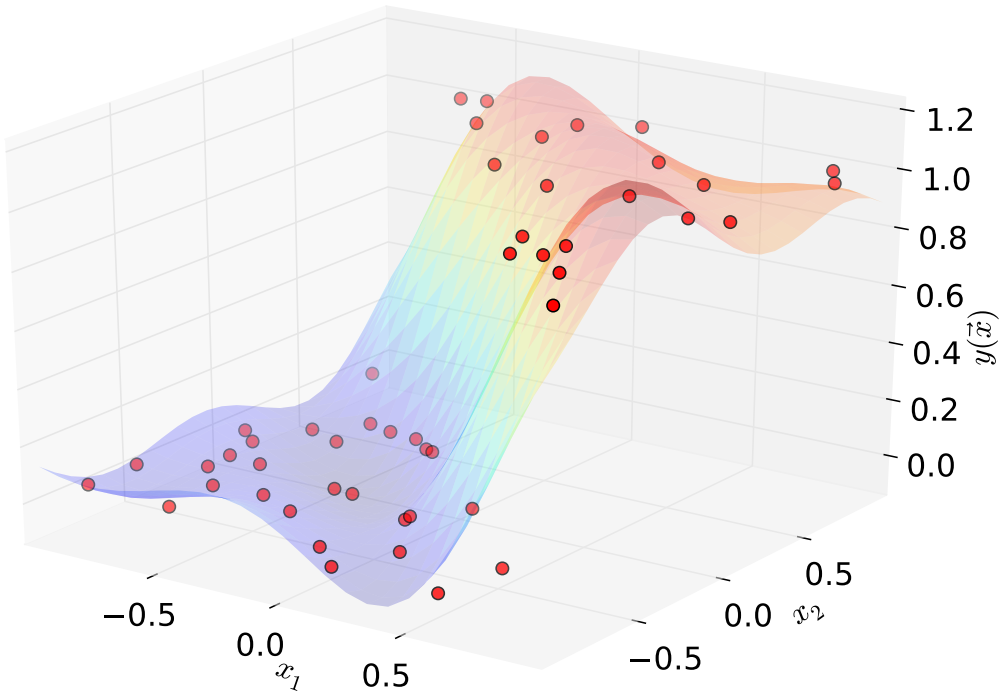


Figure 11.2: Approximation for the surrogate model constructed using default options. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

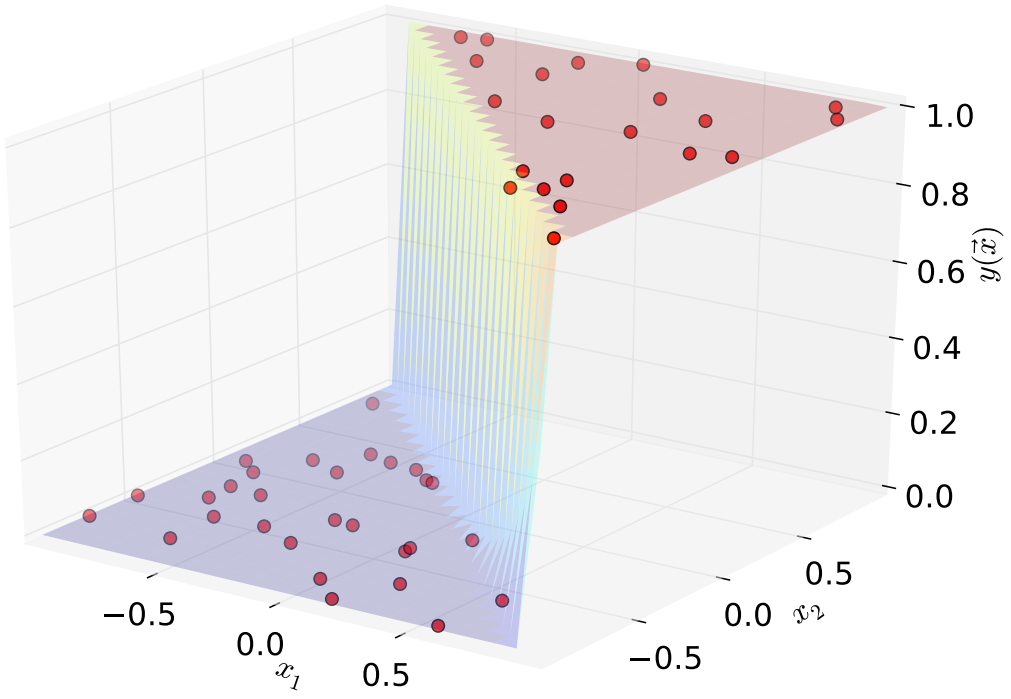


Figure 11.3: Approximation for the surrogate model constructed using recommended by post-processing function options. **Note:** This image was obtained using an older MACROS version. Actual results in the current version may differ.

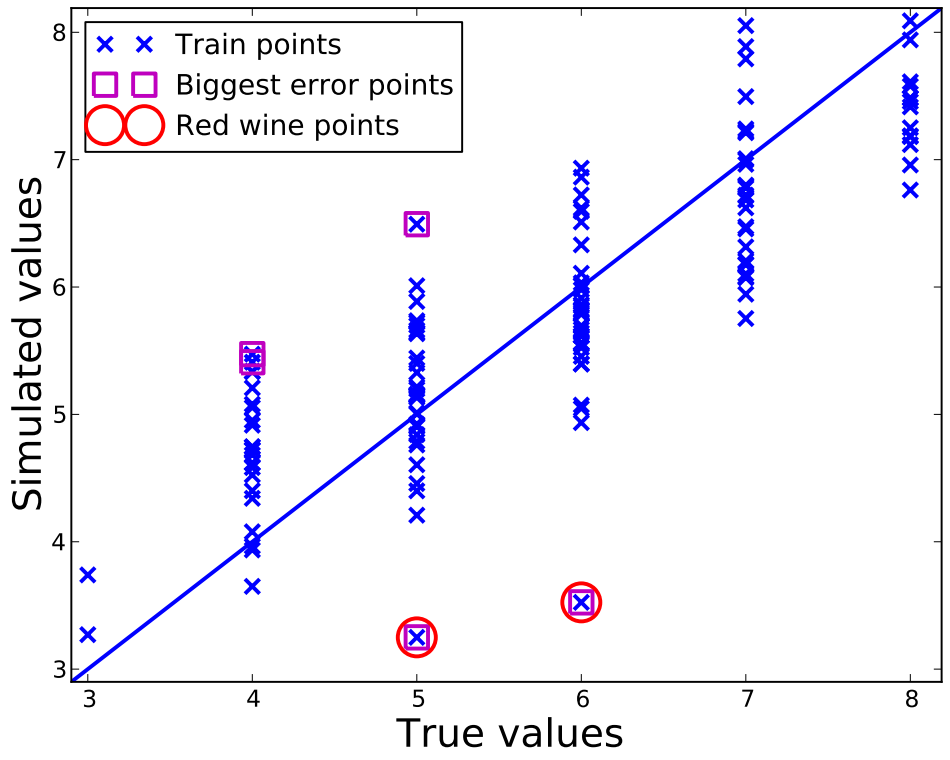


Figure 11.4: Scatter plot for the training sample for the wine quality problem

	Max	Min	RMS	MAE
Model constructed using the full training set	3.080	$1.943e-04$	0.7244	0.6008
Model constructed using the censored training set	2.323	$1.623e-04$	0.6618	0.5598

Table 11.1: Comparison of errors on the test set

Chapter 12

Tensor products of approximations

12.1 Introduction

Real-life designs of experiment often have a factored structure, in the sense that the DoE can be represented as a Cartesian product of two or more sets (*factors*). Let us denote the DoE by D and the factors by $D_1, \ldots, D_n$; then we write the factorization as

$$D = D_1 \times D_2 \times \ldots \times D_n. \tag{12.1}$$

Each factor D_k corresponds to a subset P_k of the set P of input variables $x_1, \ldots, x_{d_{\text{in}}}$, so that to the factorization corresponds a *partition*

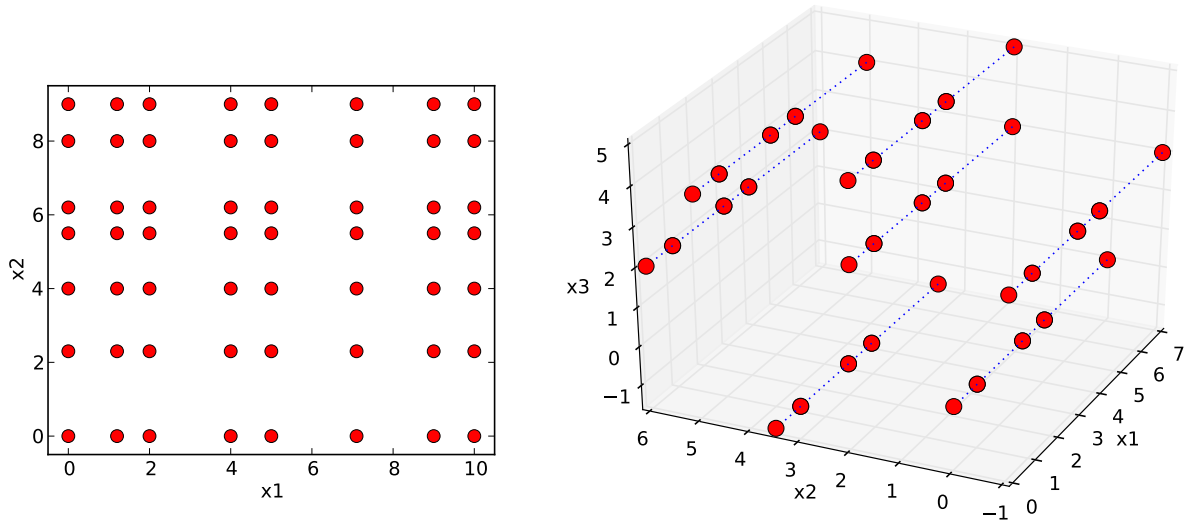
$$P = P_1 \sqcup P_2 \sqcup \ldots \sqcup P_n, \tag{12.2}$$

where by $\sqcup$ we denote the union of non-intersecting sets. We refer to the number of variables in the set P_k as the *dimension* $d_{\text{in},k}$ of the corresponding factor.

These concepts are illustrated in Figure 12.1 for the case of two factors ($n = 2$). Further details on factorization are given in Section 12.4.

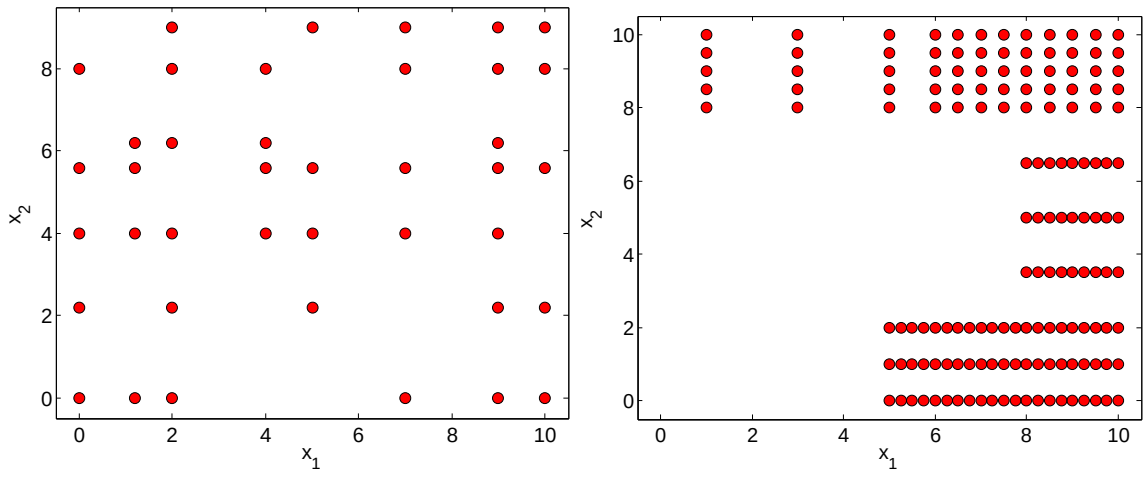
- A very common special case of factorization is the standard uniform full-factorial DoE (uniform grid). Due to the "curse of dimensionality", such a DoE is not very useful in higher dimensions, but it is very convenient and widespread in low dimensions.
- Also, in applications related to engineering or involving time series, factorization often occurs naturally when the variables are divided into two groups: one, P_1, describes spatial or temporal locations of multiple measurements performed during a single experiment (for example, spatial distribution or time evolution of some physical quantity), while another group of variables, P_2, corresponds to external conditions or parameters of the experiment. Typically, these two groups of variables are independent of each other: in all the experiments measurements are performed at the same locations. Moreover, there is often a significant anisotropy of the DOE with respect to this partition: each experiment is expensive (literally or figuratively), but once the experiment is performed, the values of the monitored quantity can be read off of multiple locations relatively easily — hence $|D_1|$ is much larger than $|D_2|$.

Another type of DOE with special structure is incomplete Cartesian product. In such case DOE D is a subset of the full product $D_1 \times D_2 \times \ldots \times D_n$. Two examples of incomplete Cartesian product are shown in figure 12.2. This type of DOE is also very common for the following reasons.



(a) **(b)**

Figure 12.1: Examples of full factorizations $D = D_1 \times D_2$:
(a) $\dim D = 2, \dim D_1 = \dim D_2 = 1$; $|D| = 56, |D_1| = 8, |D_2| = 7$; $P_1 = \{x_1\}, P_2 = \{x_2\}$.
(b) $\dim D = 3, \dim D_1 = 1, \dim D_2 = 2$; $|D| = 35, |D_1| = 5, |D_2| = 7$; $P_1 = \{x_1\}, P_2 = \{x_2, x_3\}$.
(For a better visualization, points with the common (x_2, x_3) projection are connected by dots.)



(a) **(b)**

Figure 12.2: Examples of incomplete factored set $D \subset D_1 \times D_2$:
(a) Full factored data $D_1 \times D_2$ is equal to 12.1a. An incomplete DOE D is a subset of $D_1 \times D_2$ (16 points are removed).
(b) Design D is a union of three fully factored sets which are overlap.

- Initial DOE was full product, but experiments are in progress (each calculation is time consuming operation).
- Some numerical solver was used for sample generation and it failed in few points (e.g. iterative process didn't converge).
- DOE D is a union of few full factored set. Each set corresponds to typical operating conditions of the studied object (see also figure 12.2b).
- Incomplete grid is a special case of Latin hypercube (points are selected from the full grid).

The general guidelines for the choice of DoE outlined in Section 7 are intended for generic scattered data and clearly do not apply in the above practically important and broad class of examples (or other cases of factorization), especially in the presence of anisotropy. This motivates the special treatment of the factored DoE that we describe in this chapter.

A very natural approach to approximations on a factored DoE is based on the *tensor products of approximations* constructed for individual factors. The idea of this construction for the case of full factorial DOE can be explained as follows.

First note that, thanks to factorization, for each factor D_k we can divide the training DoE D into $|D|/|D_k|$ "slices" of the form $D_k \times c$, where c is an element of the complementary product $\times_{l:l\neq k} D_l$. We can then give an alternative interpretation for the function f defined on D: namely, we can view it as a function on D_k that has multiple output components indexed by $c \in \times_{l:l\neq k} D_l$. This allows us to approximate f separately for each c using $d_{\mathrm{in},k}$-dimensional techniques. However, the drawback of this approach is that the resulting approximation is only a "partial" approximation; it is not applicable if the input variables corresponding to the factors $\times_{l:l\neq k} D_l$ have values c other than those contained in $\times_{l:l\neq k} D_l$.

The remedy for that is the tensor product of approximations over different k. For many approximation techniques, and in particular most techniques of **GTApprox**, the training process includes a fast linear phase where the approximation is expanded over dictionary (some set) of functions, and, possibly, a preceding slow nonlinear phase where the appropriate dictionary is determined (see Appendix A). For functions with multiple output components we can either choose a common dictionary for all components or choose a separate dictionary for each component – this corresponds to the `componentwise` option of **GTApprox** (see Section 6.3). Suppose that we choose a common dictionary (i.e., `componentwise` is `off`). Let us do this for each factor D_k, and then form the tensor product of the resulting n bases, i.e. form the set of vectors $\otimes_{k=1}^{n} \mathbf{v}_{k,s(k)}$, where $\mathbf{v}_{k,s(k)}$ is an element of the k-th dictionary.

In case of incomplete grids we can't divide the training DoE D into $|D|/|D_k|$ "slices" since some points are missed. It's a main reason to use explicit basis of functions instead of dictionary constructed basing on data. Such type of approximation isn't available for multidimensional techniques implemented in **GTApprox**, so we can construct tensored approximation for incomplete product only if all factors D_k are one-dimensional.

Suppose that we have a tensor product of dictionaries (or basis) Now we declare this set of functions to be the dictionary for the complete approximation of f, and it only remains to find a set of decomposition coefficients, like we do with the original approximation techniques. Computational complexity of this procedure depends on the type of DOE. In case of full factorial DOE set of linear coefficients can be found using explicit formulas in very efficient way thanks to special structure of the set. If some points are missed then such structure get broken and this explicit formulas can't be used. In spite of this we can reformulate problem

of finding optimal coefficients as optimization problem and solve it rather efficiently (also using knowledge about data structure).

Tensorial construction has several advantages. To name a few:

- It is reasonably universal: it can be used with different choices of approximation techniques in individual factors. This gives the procedure a higher degree of flexibility especially useful when tackling anisotropic data.
- Construction of the tensored approximation is relatively fast as its nonlinear phase only consists of nonlinear phases for individual factors (note that $\sum_{k=1}^{n} |D_k|$ is typically smaller than $|D| = \prod_{k=1}^{n} |D_k|$).
- In some cases, tensored approximations have properties that are hard to achieve using generic techniques for scattered data; for example, tensored splines can provide an exact fit even for very big multi-dimensional training sets, which is not possible with standard techniques (see Section 4).

At the same time, as factored data and tensored approximations have a rather special structure, they are not fully compatible with features of **GTApprox** developed with generic scattered data in mind. In particular, in **GTApprox** tensored approximations are not compatible with AE. Also, the list of approximation techniques to be tensored (see Section 12.5) is somewhat different from the list of generic techniques described in Section 4. Some of these restrictions are expected to be lifted in subsequent releases of **GTApprox**.

As a factorizable training DoE is a prerequisite for tensored approximations, **GTApprox** includes functionality to verify factorization. The user can either suggest a desired factorization to the tool or let the tool find it automatically. Also, the user can either suggest approximation techniques for individual factors or let the tool choose them automatically. These and other feature are explained in more detail in the following sections. An important topic closely related to factorization in **GTApprox** is the concept of discrete or categorical variables (see Section 12.2).

Note that algorithms for full and incomplete sets highly differ from each other so we implemented it as two different techniques: Tensor product of Approximation (`TA`) for full factored sets and incomplete Tensor product of Approximation (`iTA`) for incomplete factorization. Basically, `iTA` problem is more complicated one so this technique does not support some features available for the `TA` algorithm (like Discrete Variables). The differences between `TA` and `iTA` are described in Section 12.7. Mathematical details of this algorithms can be found in [2] and [3] (`TA` and `iTA` correspondingly).

12.2 Discrete/categorical variables

Though the purpose of approximation is to predict the behavior of the function for new values of the independent variables, in some cases some of the variables should only be considered at fixed, finitely many values. These cases correspond to *discrete* or *categorical* variables.

- A natural example of a discrete variable is some countable quantity that can only take integer values (e.g., the number of persons). In this case it typically makes no sense to ask for predictions at non-integer values.

- By a categorical variable we mean some number that is formally assigned to an object, but does not reflect any actual quantitative feature of this object. For example, we can perform an experiment under N different setups. We can introduce a variable taking values $1, 2, \ldots, N$, which formally enumerates these setups. However, this correspondence can be absolutely arbitrary: in general, the setups differ in some complex fashion and we do not expect, for example, that setup 1 is truly closer to setup 2 than to setup N, or that setup 2 is somehow truly intermediate between setups 1 and 3. It is then meaningless to ask for predictions at the values of this categorical variable other than those already present.

`TA` technique supports discrete and categorical variables if they form a factor of the DoE. In this case the tool will essentially consider the function as having several components of the output, corresponding to those values of the discrete variables that are present in the training DoE. If the constructed approximation is applied to a value of the discrete variable different from all those present in the training DoE, an error will be returned. The tool does not distinguish between discrete and categorical variables.

A variable or a set of variables can be marked as discrete using the option

`TADiscreteVariables`

(see Technical Reference for implementation details). If the marked variables do not form a factor of DoE, an error will be raised.

Alternatively, a factor can be marked as consisting of discrete variables using the option `TensorFactors` (see Section 12.4). Note that this is appropriate from the tensor product point of view, since discreteness of variables in a factor is essentially equivalent to considering the "trivial approximation" in this factor.

Note that `iTA` algorithm doesn't support Discrete variables.

12.3 The overall workflow

The overall workflow with tensored approximations is shown in Figure 12.3.

- Construction of tensored approximations is enabled by the option

  `EnableTensorFeature`

  By default, `EnableTensorFeature` is `on`: tensored approximations are enabled. Otherwise, if `EnableTensorFeature` is `off`, regardless of whether the training set can be factorized or not, the tool will use its generic approximation techniques and the decision tree described in Sections 4 and 5, in particular disregarding the product structure if present.

  Alternatively, the user can enforce tensored approximation with the option `Technique`, by setting it to `TA` (full factorial) or `iTA` (incomplete factorial). The main difference between these two methods is that the the usage of `Technique` option will produce an error if the factorization is not possible, while `EnableTensorFeature` set to `on` will create an appropriate non-tensored approximation in such case. See the decision tree 12.3 for more details.

- If tensored approximations are enabled, the tool attempts to factor the training set and to find full or incomplete product. If not successful, the tool returns to the generic techniques and decision tree (this case includes incomplete product with at least one multidimensional factor since `iTA` technique supports one-dimensional factors only). Factorization can be either performed automatically or specified by the user through the option `TensorFactors` (the latter option is valid for `TA` technique only). See Section 12.4 for more details.

- Having factored the training data, the tool constructs a set of dictionaries (or basis) of functions separately for each factor, according to the approximation technique chosen for this factor. These techniques can be either chosen automatically or specified by the user through the option `TensorFactors` (valid for `TA` only since `iTA` uses BSPL technique only). See Section 12.5 for more details.

12.4 Factorization of the training set

In this section we explain in more detail our concept of factorization and how it is handled by the tool. We will not consider the iTA technique here, since it supports incomplete products with 1-dimensional factors only (alternative factorization is not possible) and always uses the BSPL technique for a factor. The details of automatic factorization algorithm found below are related only to the case of a full factorial DoE and the TA technique usage.

From now on we suppose that all the input variables have a fixed order: $x_1, \ldots, x_{d_{\text{in}}}$. Normally, this order corresponds to the positions of the variables in the input datafile.

- In general, the tool supports arbitrary partitions (12.2) with non-empty sets P_k, in the sense that if the training DoE admits factorization (12.1) with such a partition, then the tool can construct a corresponding `TA` approximation, provided an approximation can be constructed for each factor.

- The user can directly specify a partition with the option

  `TensorFactors`

  (See Technical References for syntactic details of this command.[1]) The tool will check if the partition suggested by the user is valid, i.e., the data can be factored according to it.

- If the user did not specify a partition with the option `TensorFactors`, but tensoring is enabled by the option `EnableTensorFeature`, then the tool attempts to automatically find a factorization.

  In general, there may exist several different factorizations. For example, if a DoE of 3 variables is completely factored as $D_1 \times D_2 \times D_3$, then there also are *coarser* factorizations obtained by grouping together some of the factors, e.g. the factorization $D_1 \times D_2'$, where $D_2' = D_2 \times D_3$. In this case we say that the first factorization is *finer*. If there are two different factorizations for the given DoE, then it is not hard to see

[1] In short, the partition is specified by a list in JSON format with 0-based indexing of variables, e.g., `[ [0, 2], [1, "LR"] ]` is the partition with $P_1 = \{x_0, x_2\}, P_2 = \{x_1\}$, and with the LR technique requested for the second factor.

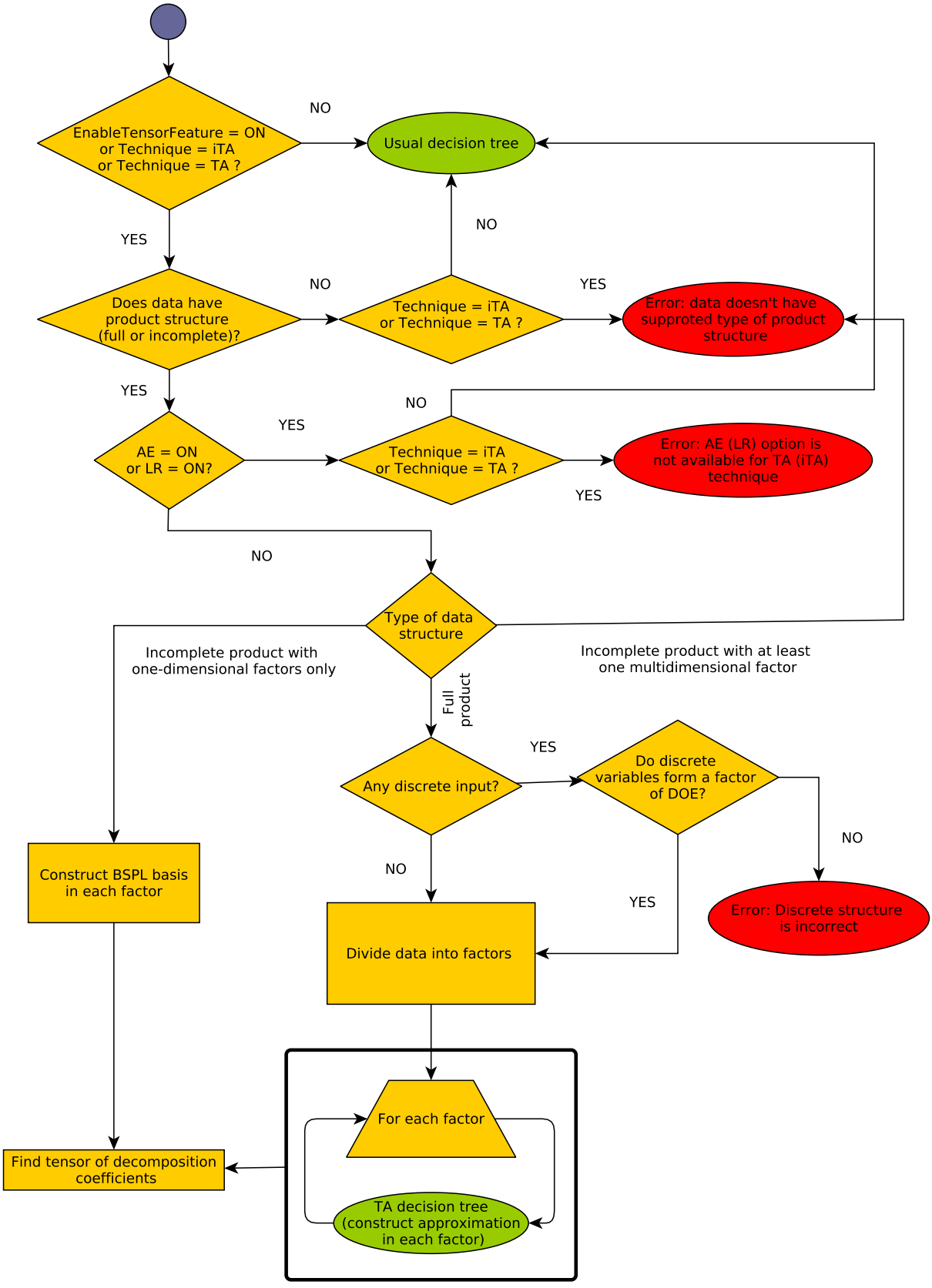


Figure 12.3: The overall workflow with tensored approximations

that there exists a factorization which is at least as fine as either of them. As a result, for any DoE there exists a unique *finest* factorization (which may of course be trivial, i.e., have only one factor).

Unfortunately, the exhaustive search for possible factorizations is computationally prohibitive, as the complexity of considering all partitions grows exponentially for large d_{in}. For this reason, the tool performs only a restricted search, which is however sufficient for typical practical purposes. We mention two main elements of the search algorithm:

- The tool attempts to factor out all subsets of a given size if their total number (determined by the dimension d_{in}) is below some threshold value. In particular, the tool is likely to find all low-dimensional factors for moderate values of d_{in}, regardless of their position among the variables $x_1, \ldots, x_{d_{\text{in}}}$.
- In addition, the tool seeks the finest factorization within *consecutive* partitions. We call a partition *consecutive* if each of its subsets consists of consecutive variables:

$$P_1 = \{x_1, \ldots, x_{m_1}\}, P_2 = \{x_{m_1+1}, \ldots, x_{x_{m_2}}\}, \ldots, P_n = \{x_{m_{n-1}+1}, \ldots, x_{d_{\text{in}}}\}$$

for a sequence $m_1 < m_2 < \ldots < m_{n-1}$. In contrast to the exponential complexity of the full search, the search over consecutive partitions is only linear in d_{in}.

12.5 Approximation in single factors

In **GTApprox**, the following approximation techniques are available for single factors of `TA` technique:

- LR0 — approximation by constant
- LR — Linear Regression; as in 4.1.1
- BSPL — cubic B-splines, similar to SPLT of Section 4.1.2; only for one-dimensional factors
- GP — Gaussian Processes; as in Section 4.1.4
- HDA — High Dimensional Approximation; as in Section 4.1.3, but with different settings
- DV — Discrete Variables; see Section 12.2.

`iTA` is based on BSPL technique only (since multidimensional factors are not supported).

Approximation techniques in `TA` may be either specified directly by the user or chosen automatically by the tool.

- *Direct specification*: If the user specifies the partition through the option `TensorFactors`, then the desired techniques may be additionally indicated for some of the factors (in the same option). See Technical References for details.

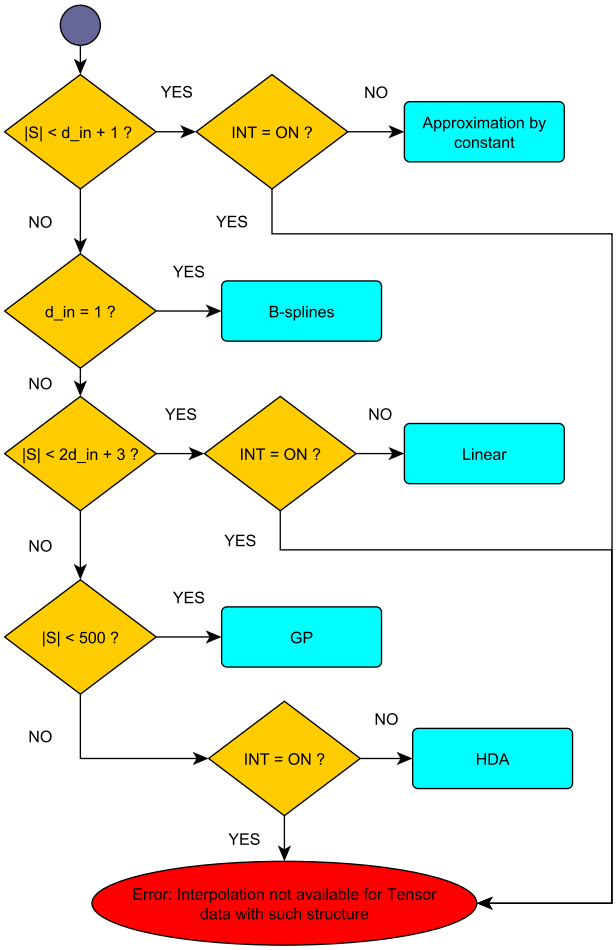


Figure 12.4: Decision tree for choosing approximation technique in a single factor

- *Automatic selection*: If for some factor the technique is not specified by the user, then it is determined automatically using the decision tree in Figure 12.4. In this decision tree, d_{in} refers to the dimension of this factor (i.e., this is $d_{\text{in},k}$ in the notation of Section 12.1), and $|S|$ refers to the size of DoE in this factor (i.e., this is $|D_k|$ in the notation of Section 12.1).

12.6 Smoothing

In **GTApprox**, tensor products of approximations can be smoothed, and their smoothing is similar to the general smoothing described in Section 6.7:

- Factors with the HDA, GP and LR techniques is smoothed in the usual way.
- Factors with the BSPL technique is smoothed in the same way as factors with HDA technique. Smoothing is based on penalization of second order derivatives (see also section 6.7).

12.7 Difference between TA and iTA techniques

The main difference between `TA` and `iTA` techniques is supported type of DOE (full and incomplete factorial correspondingly). Tensored approximation for the incomplete factorial DOE is more complicated one, so `iTA` technique has some limitations. In particular it does not support the following features:

- Discrete Variables;
- user-defined factorization (only 1-dimensional factors and BSPL technique are supported).

Also, in **GTApprox** `iTA` models are not affected by `GTApprox/ExactFitRequired` and always provide exact fit. This limitation will be removed in the subsequent releases of **GTApprox**.

12.8 Examples

In Figure 12.5 we show several different tensored approximations constructed for the same training data. In this example there are two one-dimensional factors, and the default approximation with splines in both factors looks the best. Tensored approximations with splines are usually relatively accurate and fast to construct even for large datasets; that's why they are preferred for 1D factors. For higher-dimensional factors, however, one has to use other methods, in agreement with the decision tree in Figure 12.4. As usual, the choice of the approximation type in each factor primarily depends on the amount of available data: for scarce data methods with a lower flexibility, such as LR or even LR0, are more appropriate, while for abundant data the non-linear techniques HDA and GP are more promising.

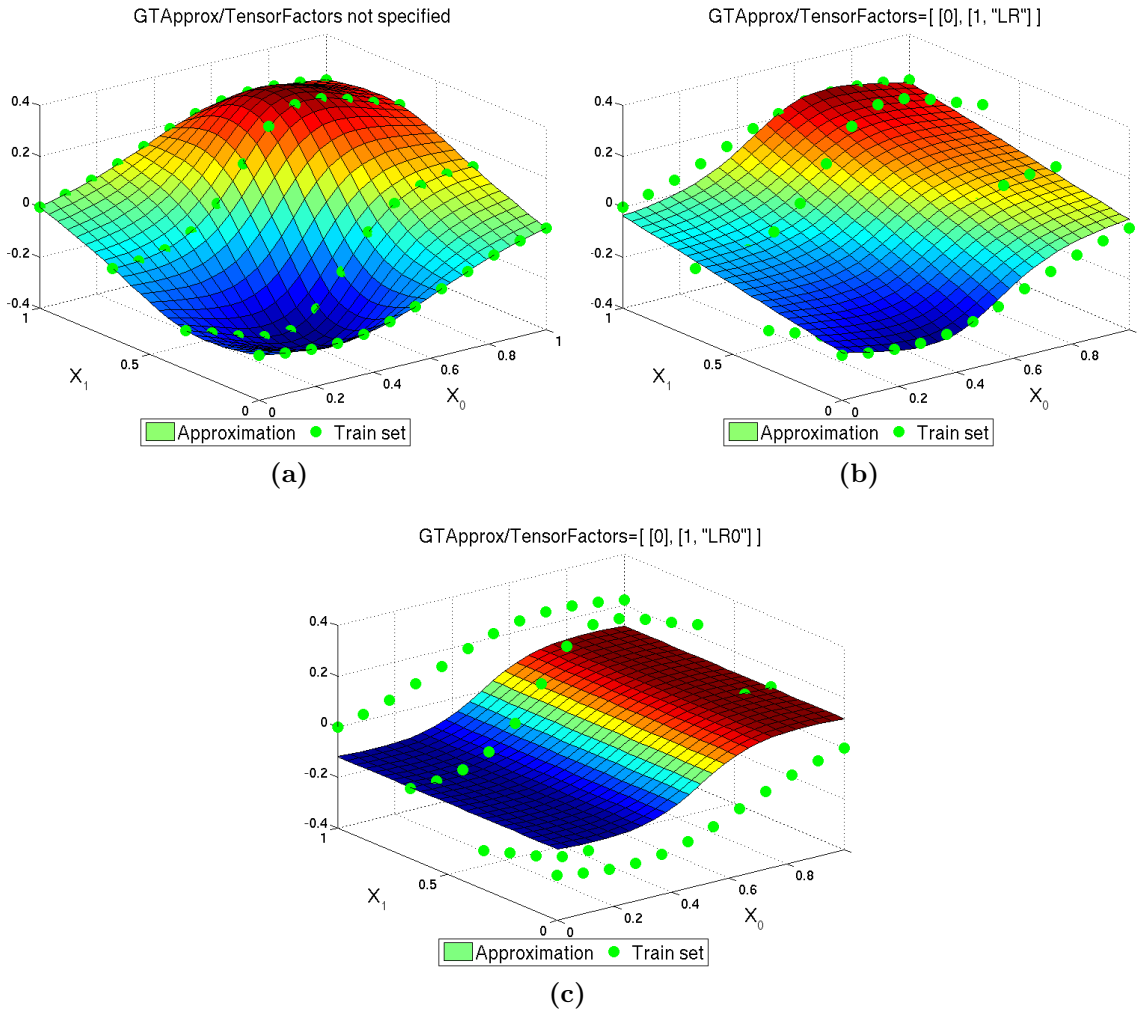


Figure 12.5: Examples of different tensored approximations for the same training set factored into two one-dimensional factors.
In (a), the default tensored approximation is used, which gives BSPL for both factors.
In (b), default approximation (BSPL) is used with $\{x_0\}$, and LR is used with $\{x_1\}$ (note the approximation is *linear* in x_1).
In (c), default approximation (BSPL) is used with $\{x_0\}$, and LR0 is used with $\{x_1\}$ (note the approximation is *constant* in x_1).
Note: This image was obtained using an older MACROS version. Actual results in the current version may differ.

Chapter 13

Multi-core scalability

13.1 Parallelization

GTApprox takes advantage of shared memory multiprocessing when multiple processor cores are available. It uses OpenMP multithreading implementation which allows user to control parallelization by setting the maximum number of threads in the parallel region. In a very general sense, certain increase in performance may be expected when using more threads, though actual **GTApprox** performance gain depends on problem properties, choice of approximation techniques, host specifications and execution environment. This section presents the conclusions drawn from **GTApprox** scalability tests and consequent recommendations for the end user.

A series of **GTApprox** techniques tests conducted internally under various supported platforms running on different hosts with different number of cores available, using a variety of samples of different dimensionality and size, showed that a significant performance increase may be achieved for HDA, GP, SGP and HDAGP techniques in case of sample size $\gtrsim 500$ by increasing the number of cores available to **GTApprox** up to 4. Further increase up to 8 cores gives little effect, with no noticeable gain after that due to parallel overhead (sometimes there is even a slight performance decrease). The nature of this dependency is the same regardless of input dimensionality, despite absolute gain values, of course, may be different.

Figure 13.1 illustrates typical behaviour of HDA technique. This particular test was run on a 3.40 GHz Intel i7-2600 CPU (4 physical cores) under Ubuntu Linux x64 with a training sample of 1000 points in 10-dimensional space. Note that due to high CPU load setting the number of OpenMP threads more than 4 on a quad-core CPU will only degrade performance (not shown), but until it does not exceed the number of physical cores, the performance scales well as expected.

The easiest way to define the number of parallel threads is by setting `OMP_NUM_THREADS` environment variable. Its default value is equal to the number of **logical** cores, which gives good results in cases described above on processors having 2-4 physical cores, but may be unwanted in other cases, especially for hyperthreading CPUs (see the next section).

13.2 Hyperthreading

GTApprox gains little from hyperthreading. In theory, it is possible to achieve some performance increase on a hyperthreading CPU compared to non-hyperthreading with same

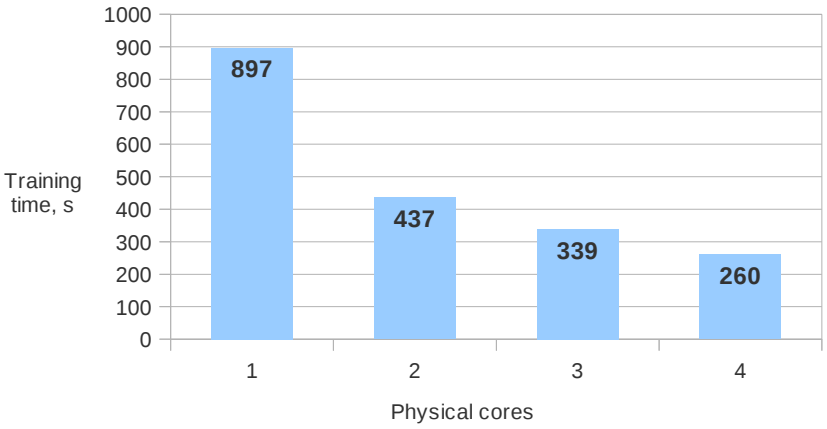


Figure 13.1: Multi-core scalability of **GTApprox** HDA. Host: Intel i7-2600 @ 3.40 GHz, Ubuntu x64. Sample size: 1000 points, dimensionality: 10.
Note: This image was obtained using an older MACROS version. Actual results in the current version may differ.

number of physical cores, if all cores are dedicated to the approximator. This, though, rarely happens in practice, and even then the gain from a virtual core can not be compared to the one from an additional physical core (i.e. quad-core with no hyperthreading is much better than dual-core hyperthreading processor even in ideal case). Moreover, often distrubuting the load to virtual cores actually decreases **GTApprox** performance, so the results are unpredictable in general.

For this reason, on hyperthreading CPUs it is recommended to keep `OMP_NUM_THREADS` not exceeding the number of available **physical** cores. However, its default value in the implementation of the OpenMP standard is equal to the number of **virtual** cores, including hyperthreaded ones. This means that **on hyperthreading CPUs user should always change the default value of** `OMP_NUM_THREADS` environment variable (see the recommended values in the next section), as not doing so may degrade **GTApprox** performance, regardless of the performed task.

13.3 Recommendations

The recommendations listed here were developed as a result of the abovementioned tests and general **GTApprox** usage on a variety of multi-core hosts. It should be noted when speaking of the desired `OMP_NUM_THREADS` setting that multiple approximators running on the same host require the recommended value to be divided by the number of simultaneously ran approximators, e.g. for two approximators on quad-core CPU the correct setting is 2, not 4 (approximators must not compete for the cores). Adjustments needed in such cases are included into the following recommendations.

- `OMP_NUM_THREADS` shall never exceed the number of physical cores divided by the number of simultaneously ran approximators. It shall also never exceed 8 even if more cores per approximator are available.

- For hyperthreading CPUs, user needs to change the default setting making it less or equal (considering other recommendations) to the number of **physical** cores divided by the number of approximators. Default is equal to the number of **logical** cores which may decrease performance.
- For LR and SPLT techniques there is no benefit in using multiple threads, so `OMP_NUM_THREADS` may be safely set to 1 to free the rest of cores for other tasks.
- The same stands for all other techniques in case of sample size $\leqslant 500$ regardless of input dimensionality — there is little to no benefit in terms of approximator performance.
- When HDA, GP, SGP or HDAGP technique is selected by user by setting the `GTApprox/Technique` option (and sample size $\lesssim 500$) or by auto selection (see Chapter 5), the most beneficial setting is 4 (or equal to number of physical processors available), divided by the number of approximators; also there is a small gain possible with up to 8 threads per approximator.

Appendix A

Mathematical details of approximation techniques implemented in GT Approx

GTApprox combines a number of approximation techniques of different types, in particular

1. Regularized linear regression (LR)
2. 1D Splines with tension (SPLT)
3. High Dimensional Approximation (HDA)
4. Gaussian Processes (GP)
5. High Dimensional Approximation combined with Gaussian Processes (HDAGP)
6. Sparse Gaussian Processes (SGP)
7. Response Surface Model (RSM)

In the present chapter short mathematical description of algorithms, realizing these methods, will be given. In the sequel we will denote by $(\mathbf{X}, \mathbf{Y})$ the training set, where $\mathbf{X} = \{X_i, i = 1, \ldots, |S|\}$ and $\mathbf{Y} = \{Y_i, i = 1, \ldots, |S|\}$.

A.1 LR

It is assumed that the training set was generated by the linear model

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\alpha} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\alpha}$ is a d_{in}-dimensional vector of unknown model parameters, $\boldsymbol{\varepsilon} \in \mathbb{R}^{|S|}$ is a vector generated by white noise process, here $\mathbf{X}$ is an extended design matrix with intercept column. Procedure, realized in GT Approx, estimates coefficients $\boldsymbol{\alpha}$ by the following formula, known as ridge regression estimate [22]

$$\hat{\boldsymbol{\alpha}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y},$$

where $\mathbf{I} \in \mathbb{R}^{d_{in} \times d_{in}}$ and regularization coefficient $\lambda \geqslant 0$ is estimated by leave-one-out cross-validation [22]. After the coefficients $\hat{\boldsymbol{\alpha}}$ are estimated, output prediction for the new input X is calculated using the following formula

$$\hat{Y} = X\hat{\boldsymbol{\alpha}}.$$

A.2 SPLT

Splines in tension is a shape-preserving generalization of spline methods for approximation of 1D functions ($d_{in} = 1$ and $d_{out} \geqslant 1$). For each interval $[X_i, X_{i+1}]$ of abscissa the tension parameter σ_i, $i = 1, \ldots, n$ is introduced. When varying the tension parameter between zero and infinity the fitting curve alters from a cubic polynomial to the linear function. The algorithm for concavity and monotonicity preserving tension factors selection results in smooth enough curve and allows to avoid oscillations in case of discontinuity in underlying function [25, 27].

A.2.1 The case of one-dimensional output.

Let us consider one-dimensional interpolation problem. Given sequence of values of abscissae $X_1 < X_2 \cdots < X_{|S|}$, and corresponding function values $Y_i, i = 1, \ldots, |S|$ the interpolation problem is to find the function $f(X)$:

$$f(X_i) = Y_i, \quad i = 1, \ldots, |S|,$$

$$f \in C^m[X_1, X_{|S|}],\ m = 1 \text{ or } 2,$$

where m is controlled by the switch `AdditionalSmoothness`.

The generalized definition of one-dimensional interpolating tension spline is the following:

$$\hat{f}(X) = \arg\min_{g(X)} \left[\int_{X_1}^{X_{|S|}} [g''(X)]^2 \, \mathrm{d}X + \sum_{k=1}^{|S|-1} \frac{\sigma_k^2}{(X_{k+1} - X_k)^2} \int_{X_k}^{X_{k+1}} [g'(X)]^2 \, \mathrm{d}X \right],$$

where $g(X_i) = Y_i$, σ_i —tension parameter in the interval $[X_i, X_{i+1}]$, $i = 1, \ldots, |S| - 1$.

The tension spline procedure chooses minimum tension factors to satisfy constraints connected with conditions on smoothness of derivatives [27]. When tension parameters are fixed the solution of the minimization problem is known [25, 27]. For convenience let us give an explicit expression for a tension spline $\hat{f}(X)$. In order to simplify the notation, we will restrict attention to the interval $[X_1, X_2]$. The following notations will be used for this explicit expression. Let Y_1, Y_2 and Y_1', Y_2' denote the data values and derivatives, respectively, associated with X_1, X_2, and define

$$h = X_2 - X_1,\ b = (X_2 - X)/h,\ s = (Y_2 - Y_1)/h,\ d_1 = s - Y_1',\ d_2 = Y_2' - s.$$

A convenient set of basis functions for the interpolant may be obtained from the following modified hyperbolic functions:

$$\mathrm{sinhm}(Z) = \sinh(Z) - Z \text{ and } \mathrm{coshm}(Z) = \cosh(Z) - 1.$$

Further, we define

$$\begin{aligned} E &= \sigma \cdot \sinh(\sigma) - 2\mathrm{coshm}(\sigma) = \mathrm{coshm}^2(\sigma) - \mathrm{sinhm}(\sigma)\sinh(\sigma), \\ \alpha_1 &= \sigma \cdot \mathrm{coshm}(\sigma) d_2 - \mathrm{sinhm}(\sigma)(d_1 + d_2), \\ \alpha_2 &= \sigma \cdot \sinh(\sigma) d_2 - \mathrm{coshm}(\sigma)(d_1 + d_2). \end{aligned}$$

For a tension parameter $\sigma > 0$ interpolant is given by the expression

$$\hat{f}(X) = Y_2 - Y_2' hb + h/(\sigma E)[\alpha_1 \mathrm{coshm}(\sigma b) - \alpha_2 \mathrm{sinhm}(\sigma b)].$$

For the tension parameter $\sigma = 0$ interpolant is given by the expression

$$\hat{f}(X) = Y_2 - h[Y_2'b + (d_1 - 2d_2)b^2 + (d_2 - d_1)b^3].$$

The tension parameters are the result of the heuristic procedure [25, 27]: for the given set of derivatives at the train points procedure computes the minimum tension parameters, required to preserve the local properties of monotonicity and convexity. If we require the resulting smoothness to be as close as possible $C^1[X_1, X_{|S|}]$ then initial values of derivatives are estimated from the data by Hymans monotonicity constrained parabolic method and only one run of tension parameter estimation procedure is necessary to obtain the solution. If the required smoothness is $C^2[X_1, X_{|S|}]$ then the iterative procedure of tension parameters and derivative estimation runs as follows:

- The initial values of tension parameters are set to zero.
- Then iteratively:
  1. Derivatives are estimated from the tridiagonal system corresponding to the continuity conditions of the second derivatives given tension parameters.
  2. Then tension parameters are estimated from the heuristic procedure given the derivative values.
- The process ends when the changes of derivative values are negligibly small.

A.2.2 The case of multidimensional output

Let us consider the case of the function with N-dimensional output $f(X) = [f_1(X), \ldots, f_N(X)]$ and the problem of this function estimation given the values of its outputs in points $X_1, X_2, \ldots, X_{|S|}$:

$$f_j(X_i) = Y_j^i, \quad i = 1, \ldots, |S|, \quad j = 1, \ldots, N.$$

Suppose that outputs belong to one class so their tension splines have the common tension parameters σ_i corresponding to the intervals $[X_i, X_{i+1}]$, $i = 1, \ldots, |S| - 1$.

We use extension of tension spline procedure to compute interpolation of the underlying function, namely spline in tension for the function with N-dimensional output solves the following minimization problem

$$[\hat{f}_1(X), \ldots, \hat{f}_N(X)] = \arg \min_{g_j : g_j(X_i) = f_j(X_i), j=1,\ldots,N} \sum_{j=1}^{N} \left\{ \sum_{k=1}^{|S|-1} \left[\int_{X_k}^{X_{k+1}} [g_j''(X)]^2 \, \mathrm{d}X + \frac{\sigma_k^2}{(X_{k+1} - X_k)^2} \int_{X_k}^{X_{k+1}} [g_j'(X)]^2 \, \mathrm{d}X \right] \right\},$$

where tension parameter σ_i in interval $[X_i, X_{i+1}]$, $i = 1, \ldots, |S| - 1$ is common for all functions.

Solution of this optimization problem is obtained in the same way as the solution of the corresponding optimization problem for the case of one-dimensional output.

A.3 HDA

HDA approximator $\hat{f}(X)$ (see also [4] for details) consists of several basic approximators $g_i(X), i = 1, 2, \ldots$, which are iteratively constructed and integrated into $\hat{f}(X)$ using specially elaborated boosting algorithm, until the accuracy of HDA approximator does not stop to increase. In fact,

$$\hat{f}(X) = \frac{1}{B}\sum_{k=1}^{B} g_k(X), \tag{A.1}$$

where the number B of basic approximators $g_k(X), k = 1, 2 \ldots, B$ are also estimated by the boosting algorithm. Each $g_k(X), k = 1, 2, \ldots$ is trained on the modified sample $(\mathbf{X}, \widetilde{\mathbf{Y}}_k)$, where $\widetilde{\mathbf{Y}}_k$ is a some function of $\mathbf{Y}$ and $\hat{\mathbf{Y}}_j = \{\hat{Y}_i = g_j(X_i), i = 1, \ldots, |S|\}, j = 1, 2, \ldots, k-1$, see [9] for details.

In turn basic approximators $g_k(X), k = 1, 2, \ldots, B$ are represented as some averages

$$g_k(X) = \frac{1}{M_k}\sum_{l=1}^{M_k} h_{k,l}(X), k = 1, 2, \ldots, B$$

of elementary approximators $h_{k,l}(X), l = 1, \ldots, M_k$, $k = 1, 2, \ldots, B$ obtained using multistart on their parameters. The value of M_k is estimated by the training algorithm of HDA. Elementary approximator model is described in the next section.

A.3.1 Elementary Approximator Model

Linear expansion in parametric functions from the dictionary is used as an elementary approximator model in HDA, i.e. the elementary approximator has the form

$$h(X) = \sum_{j=1}^{p} \alpha_j \psi_j(X), \tag{A.2}$$

where $\psi_j(X)$, $j = 1, \ldots, p$ are some parametric functions. Three main types of parametric functions are used (justification of use of such basis functions can be found in [22]), namely

1. Sigmoid basis function $\psi_j(X) = \sigma\left(\sum_{i=1}^{d_{in}} \beta_{j,i} x_i\right)$, where $X = (x_1, \ldots, x_{d_{in}})$, $\sigma(z) = \frac{e^z - 1}{e^z + 1}$. In order to model sharp features of the function $f(X)$ different parametrization can be used, namely

$$\psi_j(X) = \sigma\left(\left|\sum_{i=1}^{d_{in}} \beta_{j,i} x_i\right|^{\sigma(\alpha_j)+1} \operatorname{sign}\left(\sum_{i=1}^{d_{in}} \beta_{j,i} x_i\right)\right),$$

   where parameter α_j is adjusted independently of the parameters $\beta_j = (\beta_{j,1}, \ldots, \beta_{j,d_{in}})$. The essence is that for big negative values of α_j the function $\psi_j(X)$ behaves like step function.

2. Gaussian functions $\psi_j(X) = \exp\left(-\|X - d_j\|_2^2/\sigma_j^2\right)$.

3. Linear basis functions $\psi_j(X) = x_j$, $j = 1, 2, \ldots, d_{in}$, $X = (x_1, \ldots, x_{d_{in}})$.

Thus, the index set $J = \{1, \ldots, p\}$ can be decomposed into tree parts $J = J_{lin} \cup J_{sigmoid} \cup J_{GF}$, where $J_{lin} = \{1, \ldots, d_{in}\}$ corresponds to the linear part (linear basis functions), $J_{sigmoid}$ and J_{GF} correspond to sigmoid and Gaussian functions correspondingly. Therefore in order to fit the model (A.2) to the data we should choose the number of functions p, their type and estimate their parameters by minimization mean-square error (performance function) on the training set.

A.3.2 Training of Elementary Approximator Model

Training of Elementary approximator model (A.2) consists of the following steps:

1. Parameters of the functions from parametric dictionary are initialized, see description of an algorithm in [6]. Initial number of functions is selected with redundancy.

2. Model selection is done, i.e. values of p, $\sharp(J_{sigmoid})$ and $\sharp(J_{GF})$ are estimated ($\sharp(A)$ is a cardinality of the set A), redundant functions are deleted, see description of an algorithms in [5, 11]

3. Parameters of the approximator are tuned using Hybrid Algorithm based on Regression Analysis and gradient optimization methods. On each step of this algorithm

   - parameters of linear decomposition (A.2) are estimated using ridge regression with adaptive selection of regularization parameter, while parameters of functions from decomposition are kept fixed,
   - parameters of functions from decomposition are tuned using specially elaborated gradient method, while parameters of linear decomposition (A.2) are kept fixed [7].

   Details of the algorithm can be found in [4].

A.3.3 Correspondence between options of HDA algorithm and corresponding mathematical descriptions

Below we give the correspondence between options of HDA algorithm (described in the section 4.1.3) and mathematical variables, used for the description of the HDA algorithm in the current section.

- `HDAPhaseCount`. The number B of basic approximators (A.1), i.e. steps of the boosting algorithm, is automatically selected from the set $[1, \texttt{HDAPhaseCount}]$.

- `HDAPMin`, `HDAPMax`. The number of basis functions p, used for modeling of elementary approximator (A.2), is automatically selected from the set $[\texttt{HDAPMin}, \texttt{HDAPMax}]$.

- `HDAMultiMin`, `HDAMultiMax`. Each basic approximator (A.1) contain M elementary approximator (A.2), obtained using M multistarts. The number M is automatically selected from the set $[\texttt{HDAMultiMin}, \texttt{HDAMultiMax}]$.

- `HDAFDLinear`. This parameter controls whether linear functions are used in the decomposition (A.2) or not.

- `HDAFDSigmoid`. This parameter controls whether sigmoid functions are used in the decomposition (A.2) or not.

- `HDAFDGauss`. This parameter controls whether Gaussian functions are used in the decomposition (A.2) or not.
- `HDAInitialization`. This parameter controls the type of initialization of parameters of functions in the decomposition (A.2). In case the initialization is Random, then parameter `HDARandomSeed` can be used to define specific Seed value for the random number generator.
- `HDAHPMode`. This parameter controls, whether to use high-precision gradient algorithm for tuning of parameters of functions in the decomposition (A.2) or not.

A.4 GP

Gaussian Processes (GPs) are one of the most convenient ways to define distribution on the space of functions. GP $f(X)$ is fully determined by its mean function $m(X) = \mathbb{E}[f(X)]$ and covariance function $\mathrm{cov}(f(X), f(X')) = k(X, X') = \mathbb{E}[(f(X) - m(X))(f(X') - m(X'))]$. In the next section GP based approximation approach is described.

A.4.1 Used class of GPs

It is assumed that training data $(\mathbf{X}, \mathbf{Y})$ was generated by some GP $f(X)$

$$Y_i = Y(X_i) = f(X_i) + \varepsilon_i, i = 1, 2, \ldots, |S|,$$

where the noise ε is modeled by Gaussian white noise with zero mean and variance $\tilde{\sigma}^2$. GP based regression is interpolating, when it is assumed no noise, therefore interpolation regime of **GT Approx** is realized by setting $\tilde{\sigma}^2 \approx 0$

Also, it is assumed that GP $f(X)$ has zero mean function $m(X) = \mathbb{E}[f(X)] = 0$ and covariance function $k(X, X')$, belonging to some parametric class of covariance functions $k(X, X'|a)$, where a is a vector of unknown parameters. The following three classes of covariance functions are considered. The most common squared exponential covariance function (*weighted l_p covariance function*) has the form

$$k(X, X'|a) = \sigma^2 \exp\left(-\sum_{i=1}^{d_{in}} \theta_i^2 (x_i - x_i')^s\right), s \in [1, 2], \tag{A.3}$$

where parameters $a = \{\sigma, \theta_i, i = 1, \ldots, d_{in}\}$. *Mahalanobis covariance function* is

$$k(X, X'|a) = \sigma^2 \exp\left(-(X - X')^{\mathrm{T}} A (X - X')\right), \tag{A.4}$$

where $A \in \mathbb{R}^{d_{in} \times d_{in}}$ is a positive definite matrix and parameters $a = \{\sigma, A\}$. *The additive covariance function* of the order r has the form:

$$k(X, X'|a) = \sum_{1 \leqslant i_1 < \cdots < i_r \leqslant d_{in}} \prod_{j=1}^{r} k_{i_j}(x_{i_j}, x_{i_j}'), \tag{A.5}$$

where $k_{i_j}(x_{i_j}, x_{i_j}')$ is one-dimensional squared exponential covariance functions. Additive covariance function works especially good for high input dimensions d_{in}, and if the target function is the sum of functions which depend only on subsets of input variables [16, 17].

Under such assumptions the data sample $(\mathbf{X}, \mathbf{Y})$ is modeled by GP with zero mean and covariance function

$$\operatorname{cov}(Y(X), Y(X')) = k(X, X') + \tilde{\sigma}^2 \delta(X - X'),$$

where $\delta(X)$ is a delta function. Thus, a posteriori (with respect to the given training sample) mean value of the process for some test point X^* takes the form

$$\hat{f}(X^*) = k^* \big(K + \tilde{\sigma}^2 \mathbf{I}\big)^{-1} \mathbf{Y}, \tag{A.6}$$

where $\mathbf{I} \in \mathbb{R}^{|S| \times |S|}$ is an identity matrix, $k^* = k(X^*, \mathbf{X}) = \big[k(X^*, X_j), j = 1, \ldots, N\big]$, $K = K(\mathbf{X}, \mathbf{X}) = \big[k(X_i, X_j), i, j = 1, \ldots, N\big]$. This a posteriori mean value is used for prediction.

A posteriori (with respect to the given training sample) variance function for some test point X^* takes the form

$$\mathbb{V}\big[\hat{f}(X^*)\big] = k(X^*, X^*) + \tilde{\sigma}^2 - k^* \big(K + \tilde{\sigma}^2 \mathbf{I}\big)^{-1} (k^*)^{\mathrm{T}}. \tag{A.7}$$

This a posteriori variance is used as an accuracy evaluation of the prediction, given by a posteriori mean value.

When processing the real data, parameters of covariance function a are not known, so special tuning algorithm, described in the next section, was elaborated for their estimation.

A.4.2 Tuning of covariance function parameters

Values of covariance function parameters a are estimated using the training sample by maximizing the logarithm of corresponding likelihood, which takes the form [10, 24, 26]:

$$\log\ p(\mathbf{Y}|\mathbf{X}, a, \tilde{\sigma}) = -\frac{1}{2} \mathbf{Y}^{\mathrm{T}} (K + \tilde{\sigma}^2 \mathbf{I})^{-1} \mathbf{Y} - \frac{1}{2} \log |K + \tilde{\sigma}^2 \mathbf{I}| - \frac{|S|}{2} \log 2\pi, \tag{A.8}$$

where $|K + \tilde{\sigma}^2 \mathbf{I}|$ is a determinant of the matrix $K + \tilde{\sigma}^2 \mathbf{I}$. Besides parameters a of covariance function noise variance $\tilde{\sigma}^2$ is estimated as well by the solution of the following optimization problem

$$\log\ p(\mathbf{Y}|\mathbf{X}, a, \tilde{\sigma}) \to \max_{a, \tilde{\sigma}}. \tag{A.9}$$

For obtaining robust and well-conditioned solution of the optimization problem (A.9) special Bayesian regularization algorithm was elaborated [24].

A.4.3 Correspondence between options of GP algorithm and corresponding mathematical descriptions

Below we give the correspondence between options of GP algorithm (described in the section 4.1.4) and mathematical variables, used for the description of the GP algorithm in the current section.

- `GPPower`. This parameter defines the value of the power s, used to define the distance between input points in covariance function (A.3).
- `GPType`. This parameter defines the type of used covariance function (squared exponential covariance function (A.3), Mahalanobis covariance function (A.4) or Additive covariance function (A.5)). It should be noted that `Type` = $Mahalanobis$ can not be used with `GPPower` < 2.
- `GPInteractionCardinality`. This parameter specifies allowed values of covariance function order r, see (A.5).

A.4.4 Sparse GPs

Sparse Gaussian Processes (SGPs) are approximation of GPs. It is assumed that covariances K, K^* and $k(X, X)$ are:

$$\begin{aligned} K \approx \hat{K} = K_{S'} \cdot \left(K_{S',S'} + \sigma^2 I_M\right)^{-1} \cdot K_{S'}^T; \\ K^* \approx \hat{K}^* = K^*_{S'} \cdot \left(K_{S',S'} + \sigma^2 I_{|S'|}\right)^{-1} \cdot \left(K^*_{S'}\right)^T; \\ k(X, X) \approx \hat{k}(X, X) = K^*_{S'} \cdot \left(K_{S',S'} + \sigma^2 I_{|S'|}\right)^{-1} \cdot \left(K^*_{S'}\right)^T, \end{aligned}$$

where

- $K_{S',S'} = [k(\tilde{X}_i, \tilde{X}_j)],\ \forall k\colon \tilde{X}_k \in S',\ i, j = 1, \ldots, |S'|$ — covariance matrix for subset S',
- $K_{S'} = [k(X_i, \tilde{X}_j)],\ \forall k\colon \tilde{X}_k \in S',\ i = 1, \ldots, n,\ j = 1, \ldots, |S'|$ — cross-covariance matrix for set S and its subset S',
- $K^*_{S'} = [k(X^*, \tilde{X}_1), \ldots, k(X^*, \tilde{X}_{|S'|})]$ — covariance between new point X^* and subset S',
- I_t — unit matrix of size t.

It leads to the following equations for mean value

$$\hat{f}(X^*) = K^*_{S'} \cdot \left(K_{S',S'} + \sigma^2 I_{|S'|}\right)^{-1} K_{S'} \left(\sigma^2 I_N + K_{S'} \left(K_{S',S'} + \sigma^2 \cdot I_{|S'|}\right)^{-1} K_{S'}^T\right)^{-1} \mathbf{Y}$$

and variance

$$\mathbf{Var}[\hat{f}(X^*)] = \sigma^2 K^*_{S'} \cdot \left(\sigma^2 \left(K_{S',S'} + \sigma^2 I_{|S'|}\right) + K_{S'}^T K_{S'}\right)^{-1} \left(K^*_{S'}\right)^T + \sigma^2.$$

As in ordinary GP, when processing the real data, parameters of covariance function are not known. So parameters, estimated by GP using subset S', are used.

A.4.5 GP with errorbars

In case, if noise level in output values at points of training set is available then variances of noise these output values $(\Sigma_k = \Sigma(X_k) \in \mathbb{R}_+)_{k=1}^N$ can be used for additional regularization of GP model. The covariance function of observations becomes:

$$\mathrm{cov}(Y(X), Y(X')) = k(X, X') + \left\{\Sigma(X) + \tilde{\sigma}^2\right\}\delta(X - X'),$$

where the summand $\tilde{\sigma}^2$ is a variance of small random noise, which is used to ensure proper conditioning of GP regression problem.

The covariance matrix of training points becomes $(K + \Sigma_{\mathbf{X}} + \tilde{\sigma}^2\mathbf{I})$, where $\Sigma_{\mathbf{X}} = diag(\Sigma_1, \ldots, \Sigma_N)$ is diagonal matrix with variances of noise on the diagonal. We can easily put $(K + \Sigma_{\mathbf{X}} + \tilde{\sigma}^2\mathbf{I})$ instead of $(K + \tilde{\sigma}^2\mathbf{I})$ into formula for likelihood A.8 and solve the problem A.9 to determine values of parameters.

The formula for a posteriori (with respect to the given training sample) mean value of the process is straightforward:

$$\hat{f}(X^*) = k^*\left(K + \Sigma_{\mathbf{X}} + \tilde{\sigma}^2\mathbf{I}\right)^{-1}\mathbf{Y}, \tag{A.10}$$

where $\mathbf{I} \in \mathbb{R}^{|S|\times|S|}$ is an identity matrix, $k^* = k(X^*, \mathbf{X}) = \big[k(X^*, X_j), j = 1, \ldots, N\big]$, $K = K(\mathbf{X}, \mathbf{X}) = \big[k(X_i, X_j), i, j = 1, \ldots, N\big]$. This a posteriori mean value is used for prediction.

However, to compute a posteriori (with respect to the given training sample) variance function one need to determine value of noise variance in some new point X^*. To handle this problem it is proposed to construct special GP model $\hat{\Sigma}(X^*)$ to predict variances of noise in new point X^* on base of training set $(X_k, \Sigma_k)_{k=1}^N$. If such a model is constructed the resultant formula for a posteriori variance becomes

$$\mathbb{V}\big[\hat{f}(X^*)\big] = k(X^*, X^*) + \hat{\Sigma}(X^*) + \tilde{\sigma}^2 - k^*\big(K + \Sigma_{\mathbf{X}} + \tilde{\sigma}^2\mathbf{I}\big)^{-1}(k^*)^{\mathrm{T}}. \tag{A.11}$$

This a posteriori variance is used as an accuracy evaluation of the prediction, given by a posteriori mean value.

A.5 HDAGP

The covariance functions (A.3), (A.4) and (A.5) are stationary in the sense that they depend only on the distance between input vectors X and X', therefore, the corresponding Gaussian process is stationary and is not well suited for modeling of spatially homogeneous functions. In order to model spatially inhomogeneous functions special HDAGP model (High Dimensional Approximation combined with Gaussian Processes) was proposed [10]. According to this model the training data $(\mathbf{X}, \mathbf{Y})$ is generated by the process

$$Y_i = Y(X_i) = f(X_i) + \varepsilon_i, i = 1, 2, \ldots, |S|,$$

where the noise ε is modeled by Gaussian white noise with zero mean and variance $\tilde{\sigma}^2$, $f(X)$ is a special nonstationary Gaussian process having the form

$$f(X) = \sum_{i=1}^{p} \alpha_i \psi_i(X) + \tilde{f}(X), \tag{A.12}$$

$\tilde{f}(X)$ is a Gaussian process with zero mean and stationary covariance function $\tilde{k}(X, X'|a)$ of the form (A.3), (A.4) or (A.5), $\{\alpha_i, i = 1, \ldots, p\}$ are independent identically distributed normal random variables with zero mean and variance $\frac{\bar{\sigma}^2}{p}$, and $\big\{\psi_i(X), i = 1, \ldots, p\big\}$ is the fixed set of basis functions, obtained by applying the HDA algorithm to the training data sample.

The covariance function of Gaussian process (A.12) has the form

$$k(X, X') = \mathrm{cov}(Y(X), Y(X')) = \tilde{k}(X, X'|a) + \bar{\sigma}^2\vec{\psi}(X)^T\vec{\psi}(X') + \tilde{\sigma}^2,$$

where $\vec{\psi}(X) = \{\psi_i(X), i = 1, \ldots, p\}$.

Under the model (A.12) we can easily obtain formulas for a posteriori (with respect to the given training sample) mean value and variance for some test point X^*, analogous to formulas (A.10) and (A.11), and use them for prediction and accuracy evaluation. Tuning of unknown parameters $\{a, \bar{\sigma}, \tilde{\sigma}\}$ are done in the same way, as for GP based approximation, described in the previous section.

A.6 RSM

Let model f is defined as

$$f(X) = \alpha_0 + \sum_{i=1}^{d_{in}} \alpha_i x_i + \sum_{i,j=1}^{d_{in}} \beta_{ij} x_i x_j, \tag{A.13}$$

where $X = (x_1, \ldots, x_{d_{in}})$, α_i's and β_{ij}'s - unknown model parameters. In **GTApprox** we will refer to the model (A.13) as Response Surface Model (RSM).

It is assumed that the training set was generated by the model

$$\mathbf{Y} = f(\mathbf{X}) + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon} \in \mathbb{R}^{|S|}$ is a vector generated by white noise process and f is a Response Surface Model.

Depending on what terms in (A.13) are set to be zero, there are several types of RSM. Different RSM types are specified by option `RSMType` and summarized in Table A.1.

Type	Description
`linear`	Model contains an intercept and linear terms for each predictor (all $\beta_{ij} = 0$).
`interactions`	Model contains an intercept, linear terms, and all products of pairs of distinct input variables ($\beta_{ij} = 0$ for all $i = j$).
`purequadratic`	Model contains an intercept, linear terms, and squared terms ($\beta_{ij} = 0$ for all $i \neq j$).
`quadratic`	Model contains an intercept, linear terms, interactions, and squared terms.

Table A.1: RSM types

A.6.1 Parameters estimation

The model (A.13) can be written as $f(X) = \psi(X)\boldsymbol{c}$, where $\boldsymbol{c} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ - vector of unknown model parameters (α_i's and β_{ij}'s) and ψ - corresponding mapping. For example, $\psi(X) = (1,\ x_1, \ldots, x_{d_{in}},\ x_1x_2,\ x_1x_3, \ldots, x_{d_{in}-1}x_{d_{in}},\ x_1^2, \ldots, x_{d_{in}}^2)$ for the case of quadratic RSM. The output prediction for the new input X

$$\hat{Y} = \psi(X)\hat{\boldsymbol{c}},$$

where $\hat{\boldsymbol{c}}$ is an estimate of $\boldsymbol{c}$. Let $\boldsymbol{\Psi} = \psi(\mathbf{X})$ be the design matrix for the model (A.13).

There are a number of ways to estimate unknown coefficients $\boldsymbol{c}$. The option `RSMFeatureSelection` specifies a particular method for estimating the coefficients:

1. Least squares (`RSMFeatureSelection` = `LS`)

$$\hat{\boldsymbol{c}} = (\boldsymbol{\Psi}^T\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^T\mathbf{Y}.$$

2. Ridge regression (`RSMFeatureSelection = RidgeLS`)

$$\hat{\boldsymbol{c}} = (\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi} + \lambda\mathbf{I})^{-1}\boldsymbol{\Psi}^{\mathrm{T}}\mathbf{Y},$$

where $\mathbf{I} \in \mathbb{R}^{d_{in} \times d_{in}}$ and regularization coefficient $\lambda \geqslant 0$ is estimated by leave-one-out cross-validation [22].

3. Multiple ridge (`RSMFeatureSelection = MultipleRidge`)

$$\hat{\boldsymbol{c}} = (\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi} + \boldsymbol{\Lambda})^{-1}\boldsymbol{\Psi}^{\mathrm{T}}\mathbf{Y},$$

where $\boldsymbol{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_{d_{in}})$ - diagonal matrix of regularization parameters [5].

Regularization coefficients $\lambda_1, \lambda_2, ..., \lambda_{d_{in}}$ are estimated successively by cross-validation. Then all the regressors, coefficients of which are greater than some threshold value, are removed from the model. We assume this threshold value be equal to 10^3. Experiments showed that results of subset selection via multiple ridge are robust to the value of threshold.

4. Stepwise regression (`RSMFeatureSelection = StepwiseFit`)

Stepwise regression is ordinary least squares regression with adding and removing regressors from a model based on their statistical significance. The method begins with an initial model and then compares the explanatory power of incrementally larger and smaller models. At each step, the p-value of an F-statistic is computed to test models with and without a potential regressors [22].

The method proceeds as follows:

(a) Fit the initial model.

(b) If any regressors not in the model have p-values less than an entrance tolerance, add the one with the smallest p value and repeat this step; otherwise, go to the next step.

(c) If any regressors in the model have p-values greater than an exit tolerance, remove the one with the largest p value and go to the previous step; otherwise, end.

Entrance and exit tolerances are specified by options `RSMStepwiseFit/penter` and `RSMStepwiseFit/premove` accordingly.

In **GTApprox** two approaches for Stepwise regression are realized:

(a) Forward selection, which involves starting with no variables in the model, testing the addition of each variable, adding the variable that improves the model the most, and repeating this process until none improves the model.

(b) Backward elimination, which involves starting with all candidate variables, testing the deletion of each variable, deleting the variable that improves the model the most by being deleted, and repeating this process until no further improvement is possible.

Depending on the terms included in the initial model and the order in which terms are moved in and out, the method may build different models from the same set of potential terms.

Stepwise regression type is specified by the option `RSMStepwiseFit/inmodel`. We have `RSMStepwiseFit/inmodel` = `ExcludeAll` for Forward selection and `RSMStepwiseFit/inmodel` = `IncludeAll` for Backward elimination.

Note that Regularized linear regression in **GTApprox** (see 4.1.1) is a special case of RSM for `RSMType` = `linear`, `RSMFeatureSelection` = `RidgeLS`.

A.6.2 Categorical variables

GTApprox RSM allows to process *categorical variables* that is variables that can take a limited numbers of possible values. In order to use categorical variables in the regression model they are automatically pre-coded with so-called *dummy variables*. Dummy variables takes only the values 0 or 1 and it takes $k-1$ dummy variables to encode the categorical variable with k different values. Shown in table A.2 is an example of such a pre-coding for the categorical variable with 4 possible values.

The list of categorical variables is specified by option `RSMCategoricalVariables`.

C	D_1	D_2	D_3
1	0	0	0
2	1	0	0
3	0	1	0
4	0	0	1

Table A.2: Pre-coding with dummy variables

Bibliography

[1] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

[2] M. Belyaev. Approximation problem for cartesian product based data. *Proceedings of MIPT*, 5(3):11–23, 2013.

[3] M. Belyaev. Approximation problem for factorized data. *Artificial intelligence and decision theory*, 3:95–110, 2013.

[4] M. Belyaev and E. Burnaev. Approximation of a multidimensional dependency based on a linear expansion in a dictionary of parametric functions. *Informatics and its Applications*, 7(3), 2013.

[5] M. Belyaev, E. Burnaev, and A. Lyubin. Parametric dictionary selection for approximation of multidimensional dependency. In *Proceedings of All-Russian conference "Mathematical methods of pattern recognition (MMPR-15)"*, pages 146–150, Petrozavodsk, Russia, 11–17 September 2011.

[6] M. Belyaev, E. Burnaev, P. Yerofeyev, and P. Prikhodko. Methods of nonlinear regression models initialization. In *Proceedings of All-Russian conference "Mathematical methods of pattern recognition (MMPR-15)"*, pages 150–153, Petrozavodsk, Russia, 2011.

[7] M. Belyaev and A. Lyubin. Some peculiarities of optimization problem arising in construction of multidimensional approximation. In *Proceedings of the conference "Information Technology and Systems – 2011"*, pages 415–422, Gelendzhik, Russia, 2011.

[8] A. Bernstein, M. Belyaev, E. Burnaev, and Y. Yanovich. Smoothing of surrogate models. In *Proceedings of the conference "Information Technology and Systems – 2011"*, pages 423–432, Gelendzhik, Russia, 2011.

[9] E. Burnaev and P. Prikhodko. Theoretical properties of procedure for construction of regression ensemble based on bagging and boosting. In *Proceedings of the conference "Information Technology and Systems – 2011"*, pages 438–443, Gelendzhik, Russia, 2011.

[10] E. Burnaev, A. Zaytsev, M. Panov, P. Prihodko, and Y. Yanovich. Modeling of nonstationary covariance function of gaussian process using decomposition in dictionary of nonlinear functions. In *Proceedings of the conference "Information Technology and Systems – 2011"*, pages 357–362, Gelendzhik, Russia, 2011.

[11] E. V. Burnaev, M. G. Belyaev, and P. V. Prihodko. About hybrid algorithm for tuning of parameters in approximation based on linear expansions in parametric functions. In *Proceeding of the 8th International Conference "Intelligent Information Processing"*, pages 200–203, Cyprus, 2010.

[12] R. J. W. Chen and A. Sudjianto. On sequential sampling for global metamodeling in engineering design. In *Proceedings of DETC02 ASME 2002 Design Engineering Technical Conferences And Computers and Information in Engineering Conference*. ASME, 2002.

[13] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[14] N. A. C. Cressie. *Statistics for Spatial Data*. Wiley, 1993.

[15] DATADVANCE, llc. *MACROS Generic Tool for Approximation: Technical Reference*, 2011.

[16] N. Durrande, D. Ginsbourger, and O. Roustant. Additive kernels for gaussian process modeling. *arXiv preprint arXiv:1103.4023*, 2011.

[17] D. Duvenaud, H. Nickisch, and C. E. Rasmussen. Additive gaussian processes. *arXiv preprint arXiv:1112.4394*, 2011.

[18] B. S. Everitt. *The Cambridge Dictionary of Statistics*. Cambridge University Press, 2002.

[19] A. Forrester, A. Sóbester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. Progress in astronautics and aeronautics. J. Wiley, 2008.

[20] L. Foster, A. Waagen, and N. Aijaz. Stable and efficient gaussian process calculations. *Journal of machine learning*, 10:857–882, 2009.

[21] S. Geisser. *Predictive Inference*. New York: Chapman and Hall, 1993.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2008.

[23] J. M. Hyman. Accurate monotonicity preserving cubic interpolation. *SIAM J. Sci. Stat. Comput.*, 4(4):645–654, 1983.

[24] M. E. Panov, E. V. Burnaev, and A. A. Zaytsev. Bayesian regularization for regression based on gaussian processes. In *Proceedings of All-Russian conference "Mathematical methods of pattern recognition (MMPR-15)"*, pages 142–146, Petrozavodsk, Russia, 11–17 September 2011.

[25] S. Pruess. An algorithm for computing smoothing splines in tension. *Computing*, 19:365–373, 1977.

[26] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[27] R. J. Renka. Interpolatory tension splines with automatic selection of tension factors. *Society for Industrial and Applied Mathematics, Journal for Scientific and Statistical Computing*, 8:393–415, 1987.

[28] P. Rentrop. An algorithm for the computation of the exponential spline. *Numerische Mathematik*, 35:81–93, 1980.

[29] C. Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik*, 46:224–243, 1901.

[30] T. J. Santner, B. Williams, and W. Notz. *The Design and Analysis of Computer Experiments*. Springer-Verlag, 2003.

[31] I. V. Tetko, D. J. Livingstone, and A. I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *J. Chem. Inf. Comput. Sci.*, 35(5):826–833, 1995.

[32] D. Yarotsky. HDA/AE development plan: general functionality and user interface. Technical report, DATADVANCE, 2011.

Acronyms

AE Accuracy Evaluation. 5

ExFit Exact fit mode. 5

GT Approx Generic Tool for Approximation. 3

GT DoE Generic Tool for DoE. 40

IV Internal Validation. 5

Int Exact interpolation mode (synonym for exact fit). 5

Lin Linear mode. 5

DoE Design of Experiment. 3

GP Gaussian Processes. 16

HDA High Dimensional Approximation. 11

HDAGP High Dimensional Approximation combined with Gaussian Processes. 19

LR Linear Regression. 10

MAE Mean Absolute Error. 38

RMS root-mean-squared error. 38

RSM Response Surface Model. 21

SGP Sparse Gaussian Processes. 20

SPLT 1D Splines with tension. 10

Index: Concepts

accuracy, 38
Accuracy Evaluation, 48
AE, 48
approximation, 3
approximation techniques, 4, 10

categorical variable, 76
consecutive partition, 80

decision tree, 26
design of experiment, 3
design space, 4
dimension of a factor, 73
discrete variable, 76
DoE, 3

effective input dimension, 9
effective sample size, 9
errobar, 35

factor, 73

heteroscedasticity, 34

Internal Validation, 57
interpolation, 4, 31
IV, 57

model, 5

noisy problems, 33

overfitting, 39
overtraining, 39

partition, 73
predictive power, 38

response function, 3

size of the training set, 3
smoothing, 4, 36

tensor product, 75
test set, 38
training data/set/sample, 3

Index: Options

Note: these are shortened names of the options. The full names are obtained by adding `GTApprox/` in front of the shortened name, e.g. `GTApprox/Technique`.

`Accelerator`, 44
`AccuracyEvaluation`, 48
`Componentwise`, 33
`EnableTensorFeature`, 77
`GPInteractionCardinality`, 19, 93
`GPLinearTrend`, 18
`GPMeanValue`, 19
`GPPower`, 18, 93
`GPType`, 18, 93
`HDAFDGauss`, 15
`HDAFDLinear`, 14
`HDAFDSigmoid`, 15
`HDAHessianReduction`, 16
`HDAInitialization`, 15
`HDAMultiMax`, 14
`HDAMultiMin`, 14
`HDAPhaseCount`, 13
`HDAPMax`, 13
`HDAPMin`, 13
`HDARandomSeed`, 16
`Heteroscedastic`, 35
`InternalValidation`, 57
`InterpolationRequired`, 31
`IVRandomSeed`, 57
`IVSubsetCount`, 57
`IVTrainingCount`, 57
`LinearityRequired`, 30
`RSMCategoricalVariables`, 23
`RSMFeatureSelection`, 22
`RSMMapping`, 22
`RSMStepwiseFit/inmodel`, 22
`RSMStepwiseFit/penter`, 23
`RSMStepwiseFit/premove`, 23
`RSMType`, 22
`SGPNumberOfBasePoints`, 21
`SGPSeedValue`, 21
`SPLTContinuity`, 11
`TADiscreteVariables`, 77
`Technique`, 26
`TensorFactors`, 78